

APPENDIX B

HDL-Parser.pm

```

use vpp;
#####
5 # these files should really be in some util module
sub HDL_Display_Hash
{
    my (%h) = @_;

10    goldfish "displaying hash";
    foreach $a (keys (%h))
    {
        warn "$a -> $h{$a}\n";
    }
15 }

sub HDL_Read_File
{
    my $file = shift or ribbit "no file specified\n";
20    open (FILE,"<$file") or ribbit "cannot open file ($file) ($!)\n";
    my $return_string;
    while (<FILE>)
    {
        $return_string .= $_;
25    }
    close (FILE);
    return($return_string);
}

30 sub HDL_Write_File
{
    my $file = shift or ribbit "no file specified\n";
    my $string = shift or ribbit "no string specified\n";

35    open (FILE,">$file") or ribbit "cannot open file ($file) ($!)\n";
    print FILE $string;
    close (FILE);
}
# these files should really be in some util module
40 #####
#####
# HDL_Remove_Comments
#
45 # removes comments from an entire string depending on language
# specified
#####
sub HDL_Remove_Comments
{
50    my $string = shift or ribbit "no string specified\n";
    my $language = shift or "verilog";

    if ($language =~ /verilog/i)
    {
55        $string =~ s|\\\/\*.*?\*\\\/||gs;
        $string =~ s|\\\/\*.*$||gm;
        return ($string);
    }
    if ($language =~ /vhdl/i)
60    {
        $string =~ s|\\\/\*.*$||gm;
        return ($string);
    }
}

```

09830106 061201

ribbit "language (language) not understood\n";

}

#

5 # HDL_Count_Parentheses

#

so i have a string always @(blow(me)(leonardo|synplicity)) blerg (boof).

I want to perform computations on the string surrounded by the

beginning and last parentheses. I call HDL_Count_Parentheses and it

10 # returns 3 values, the beginning string: "always @", the parenthesized string

"blow(me)(leonardo|synplicity)" and the last string "blerg (boof)".

#

If I want to search on something other than parentheses, say begin,end, I can

place their values in \$begin_match and \$end_match.

15 #####

#

sub HDL_Count_Parentheses

{

20 my (\$string,\$begin_match,\$end_match) = @_;

my \$begin_string;

my \$paren_string;

my \$end_string;

my \$begin_match_default = '\s*\(\s*';

my \$end_match_default = '\s*\)\s*';

25 \$begin_match = \$begin_match_default unless (\$begin_match);

\$end_match = \$end_match_default unless (\$end_match);

return("", "", "\$string")

30 unless (\$string =~ /^(.*)\$begin_match(.*)\$/s);

\$begin_string = \$1;

my \$paren_count = 1;

\$end_string = \$2;

35 while (\$end_string =~ s/^(.*)(\$begin_match|\$end_match)(.*)\$/3/s)

{

my \$match;

\$match = \$2;

40 \$paren_string .= \$1;

if (\$match =~ /\$begin_match/)

{

\$paren_count++;

45 }

else

{

\$paren_count = \$paren_count - 1;

}

50 last if (\$paren_count == 0);

#else

\$paren_string .= \$match;

}

55 ribbit "mismatched \$begin_match,\$end_match in string

\$begin_string\$paren_string\$end_string" if (\$paren_count != 0);

return (\$begin_string,\$paren_string,\$end_string);

60 }

#####

HDL_Get_Module_Info

09330105-061201


```

#
# HDL_Get_Module_Info goes through a string, finds all the modules it
# can and gets as much information as it can about each module. It
# stores all of the information in a gigantic hash pointed to by $mp.
5 # Here is the structure of $mp. If something in the table below has
# <> around it i.e <foo_name>. That means the name is a variable. The
# perl type cast operators are used to specify what type the value is
#
# %mp
10 # %<module name> the name of each module found
# @port_order the order of ports declared by <module name>
# %signal list of ports and signals
# %<port or signal name> name of port/signal
# $left port/signal left index
15 # $right port/signal right index
# $width port/signal width i.e (abs ($left - $right)
+ 1)
# $direction port only (input/output/inout)
# $type wire or register (verilog) (type for vhd1
20 e.g. std_logic_vector)
# $assignment the value assigned to signal, or the entire
always block
# for registers
# %instance list of all instances in module
25 # %<instance_name> name of instance
# $module module that is being instantiated
# $library vhd1 only (name of library)
# %hash points to $mp{$module} gets assigned in
HDL_Munge_Data
30 # %connection list of connections
# %<module_port> name of signal that points to
$module.<module_port>
# @always_order array of each always blocks from "begin" to
"end"
35 #
#####

sub HDL_Get_Module_Info
{
40 my $string = shift or ribbit "no string ($string) specified\n";
my $language = shift or "verilog";

my %module;
my $mp = shift;
45 $mp = \%module unless $mp;

$string = &HDL_Remove_Comments ($string,$language);
if ($language =~ /verilog/i)
{
50 # crush definitions for now. Later we can use v2vhd's reader
# that handles defines

$string =~ s/^\s*\`define\s+.*$//mg;
#suck up an entire module declaration.
55 while ($string =~ s/\bmodule\s+(\w+)\s* #module <name>
\((.*)\s* #(<port_list>);
(.*) #<module_innards>
\bendmodule\b//sx) #endmodule

{
60 my ($name,$port_list,$module_innards) = ($1,$2,$3);

#####
#put cleaned up port list into hash

```

```

$port_list = s/^s*(.*?)\s*$/1/s;
@{$mp->{$name}{port_order}} = split (/s*\s*,\s*/s,$port_list);

#####
5 # now go through each command in module innards and extract data
my $port_types = "input\|output\|inout";
#foreach $command (split (/s*\s*,\s*/s,$module_innards))
while ($module_innards =~ s/^s*(.*?)\s*//s)
10 {
    # $command =~ s/^s*(.*?)\s*$/1/s;
    my $command = $1;
    # ports and signal declarations
    if ($command =~ /^s*($port_types|reg|wire|integer)([\s\[].*)/os)
15 {
        my ($type,$ports) = ($1,$2);
        my $width;

        if ($ports =~ s/^s*\[?(.*?)\:(.*?)\](.*)/$3/)
20 {
            # if port [left:right] store vector fields
            # appropriately

            my ($left_index,$right_index) = ($1,$2);
            $width = abs ($left_index - $right_index) + 1;
            $mp->{$name}{signal}{$port}{left} = $left_index;
            $mp->{$name}{signal}{$port}{right} = $right_index;
        }
        else
30 {
            # else its just a bit of width 1.
            $width = 1;
        }

        # ports/signals can be comma separated, so loop through
        # all comma separated ports
        $ports =~ s/^s*(.*?)\s*$/1/s;
        foreach $port (split(/s*\s*,\s*/s,$ports))
40 {
            if ($type =~ /$port_types/o)
            {
                #if this was a port declaration...
                #check that $port was defined in port list above.
                ribbit
45 "port $port is not found in port list $ports\n"
                unless ($port_list
s/\s*\b$port\b\s*\s*,\s*/s);

                #store additional data. type may get overwritten
                #later if the same signal is declared as a reg
                $mp->{$name}{signal}{$port}{direction} =
                    $type;
                $mp->{$name}{signal}{$port}{type} = "wire";
            }
            else
55 {
                #not a port declaration
                $mp->{$name}{signal}{$port}{type} = $type;

                #only wire statements can have =s in them.
                if ($ports =~ s/^.*?\s*=\s*(.*?)\s*//s)
60 {
                    $mp->{$name}{signal}{$port}{assignment} = $1;

```

09880106 "061201

```

    }
    }
    $mp->{$name}{signal}{$port}{width} = $width;
}
next;
}

#now parse instantiations.
if ($command =~ /^s*(\w+)                #<module_name>
    \s*(\w+)                #<instantiation_name>
    \s*(\s*(.*?)\s*\)      #((.a) b, (.c) d)
    \s*/sx)
{
    my $instance_name = $2;
    $mp->{$name}{instance}{$instance_name}{module} = $1;
    my $connection_list = $3;

    #make a ref for easier coding later ref comes into
    #existence after we declare {module} = <module_name>

    my $ref = $mp->{$name}{instance}{$instance_name};
    #store away connection info
    foreach $connection (split
        (/s*\s*,\s*/s,$connection_list)
        )
    {
        ($connection =~ /^s*\s*(\w+)\s*(\s*(.*?)\s*\)/) or ribbit
            "connection ($connection) not understood\n";
        my ($module_port,$external_port) = ($1,$2);
        $ref->{connection}{$module_port} = $external_port;
    }
}

#assign statements
while ($command =~
s/^s*(assign\s+[\^=]*?) ([a-zA-Z]\w*) ([\^=]*?\s*(.*?)\s*)$
/$1\ $3/sx)
{
    $mp->{$name}{signal}{$2}{assignment} = $4;
}

#####
# handle always statements
# there is currently a screw case here. if you do
# something like always if adfsdf; else case asdfsadffa;,
# it won't work. if you do always () a <= c; or always ()
# begin .... end it will work.
if ($command =~ /\b(always.*?(\\bbegin)?)/s)
{
    my $pre_begin = $1;
    my ($pre_begin,$always_guts,$after_guts) =
&HDL_Count_Parentheses("$command\;$module_innards","begin","end");

    #print "command ($command) pb ($pre_begin) ag ($always_guts) afg
($after_guts)\n";
    if ($always_guts)
    {
        $module_innards = $after_guts;
        $always_guts = "$pre_begin begin $always_guts end";
    }
    else

```

```

    {
        $always_guts = $command;
    }
    push (@{$mp->{$name}{always_order}}, $always_guts);
5
    #####
    # search through a always statement. Whenever an
    # assignment is made to <signal> set <signal>{assignment} =
    # the whole always statement.
10    my $tmp_always_guts = $always_guts;
    while ($tmp_always_guts =~
s/(\sbegin\s|;)\s*(\w+)\s*<?=(\{.*?\};/$1/is)
    {
        $mp->{$name}{signal}{$2}{assignment} = $always_guts;
15    }
    }

}

20    # now we are done with module innards, make sure all ports
    # were defined.
    if ($port_list =~ /(\w+)/)
    {
        ribbit "module $name has ports ($port_list) specified in port list,
25        but not declared inside module declaration\n";
    }
}
return ($mp);
}
30    if ($language =~ /vhdl/i)
    {
        $string =~ tr/A-Z/a-z/;
35        #now everything is lower case.

        while ($string =~
s/\s*\bENTITY\s+(\w+)\s+IS\s+(.*?)\s+END\s+\s*\s*;/si)
        {
40            my ($name) = $1;
            my ($port,$port_list,$sc) = &HDL_Count_Parentheses($2);
            ribbit "Entity $module_name declaration, port list not understood
($port) ($sc)"
45            unless (($port =~ /^port\s*$/is) &&
                ($sc =~ /\s*$/is));

            $port_list =~ s/^\s*(.*?)\s*$/1/s;
            foreach $port_declaration (split (/s*\s*;/s,$port_list))
            {
50                ($port_declaration =~ s/^\s*SIGNAL\s+(\w+) #port_name
                    \s*:\s*(\w+) #direction
                    \s+(\w+) #type
                    //six) or ribbit
                    "entity $module_name, port $port misunderstood";
55                my ($port,
                    $direction,
                    $type) = ($1,$2,$3);

                die "port is bogus ($port,$direction,$type)\n"
60                . "($1,$2,$3) ($port_declaration)\n"
                    if ($port eq "");

                push (@{$mp->{$module_name}{port_order}}, $port_name);

```

```

$direction .= "put"
    unless ($direction eq "inout");
$mp->{$name}{signal}{$port}{direction} =
    $direction;
5   $mp->{$name}{signal}{$port}{type} =
    $type ;

my $width;
10  if ($port_declaration =~ s/\((.*)\)\//s)
    {
        my $vector = $1;
        my ($left_index,$foo,$right_index) =
            ($vector =~ s/^\s*(.*?)\s*(down)?to\s*(.*)\s*$//si)
            or ribbit "port $port, vector $vector not understood";

15         $width = abs ($left_index - $right_index) + 1;
        $mp->{$name}{signal}{$port}{left} = $left_index;
        $mp->{$name}{signal}{$port}{right} = $right_index;
    }
20  else
    {
        $width = 1;
    }
    $mp->{$name}{signal}{$port}{width} = $width;
25  }
}

#now get architecture
while ($string =~ s/\b
30  architecture\s+          # architecture
    (\w+)\s+                 # <behavior>
    of\s+(\w+)\s+            # of <entity>
    is\s+(.*?)\b             # is <signal_list>
    begin\b                  # begin
35  (.*?)\b                   # (behavior description)
    end\s+\\s*\;;\s*//six)    # end <behavior> ;
    {
        my ($name,$signal_list,$behavior) = ($2,$3,$4);
40        $signal_list =~ s/^\s*(.*?)\s*$/$1/s;

        foreach $signal_declaration (split (/\\s*\;;\s*/s,$signal_list))
        {
            ($signal_declaration =~
45            s/^\s*(SIGNAL|SHARED\s+VARIABLE)\s+(\w+)    # SIGNAL <signal_name>
            \s*:\s*(\w+)                                  # : <type>
            //six) or next;                                # (forget about vhd1

                                                                # declarations for now.)

50            my ($signal, $type) = ($2,$3);

            # type e.g. is std_logic or std_logic_vector
            $mp->{$name}{signal}{$signal}{type} = $type;
55            my $width;
            #####
            #signal_declaration now has only vector information left
            #(if anything).
60            if ($signal_declaration =~ s/\((.*)\)\//s)
                {
                    #assign vector info if it is a vector.
                    my $vector = $1;

```

09830106.061201

TYPE

```

my ($left_index,$foo,$right_index) =
    ($vector =~ s/^(.*?)\s*(down)?to\s*(.*)$//si)
    or ribbit "port $port, vector $vector not understood";

5      $width = abs ($left_index - $right_index) + 1;
      $mp->{$name}{signal}{$signal}{left} = $left_index;
      $mp->{$name}{signal}{$signal}{right} = $right_index;
    }
    else
10     {
        $width = 1;
    }
    $mp->{$name}{signal}{$signal}{width} = $width;
}

15 $behavior =~ s/^\s*(.*?)\s*$/1/s;

#####
# handle process statements
20 while ($behavior =~ s/\bprocess\b\s*      #process
    (.*?)                                  #stuff
    \s*\bend\s+process\s*\;;\s*//six) #end process
{
    my $process_guts = $1;
    push (@{$mp->{$name}{always_order}}, $process_guts);

    #####
    # search through a process statement. Whenever an
    # assignment is made to <signal> set <signal>{assignment} =
    # the whole process statement.
    my $tmp_process_guts = $process_guts;
    while ($tmp_process_guts =~ s/(\bTHEN\s|\\)\s*(\w+)\s*(\<|\\:)\s*(.*?)\s*;/1/is) ==
35     {
        $mp->{$name}{signal}{$2}{assignment} = $process_guts;
    }
}

#####
40 #now all that is left is signal assignments and instantiation
foreach $command (split (/\\s*\\;\\s*/,$behavior))
{
    # just handle simple wire assignments, no lhs concatenation
    # craziness for now.
    if ($command =~ /^\\s*(\\w+)\\s*      #<lhs>
        \\<\\s*      #<= (assignment operator)
        (.*?)\\s*$      #<rhs>
        /six)
    {
50         my ($lhs,$rhs) = ($1,$2);
        ribbit "lhs is null in c ($command)"
            if ($lhs eq "");
        ribbit "rhs is null in c ($command)"
            if ($rhs eq "");

55         $mp->{$name}{signal}{$lhs}{assignment} = $rhs;
        next;
    }

60     #handle instantiation
    if ($command =~
        /^(\\w+)\\s*\\:\\s*(entity\\s+)?      #<instantiation> : entity
        (\\w+)\\.?(\\w*)\\s*                  #<lib>.<module>

```

09830106 "061201
T02T90" 9070990

09630106"061201

```
(.*)ix)                                #<port map (clk...)
{
5    my ($instance_name,
        $entity,
        $library_or_module,
        $module,
        $rest)
        = ($1,$2,$3,$4,$5);

10    #####
    # if "entity" is declared, then module has a library,
    # otherwise its a component with no library.
    my $library = "";
    if ($entity)
15    {
        $library = $library_or_module;
    }
    else
    {
20        $module = $library_or_module;
    }

    #assign stuff to hash
    $mp->{$name}{instance}{$instance_name}{module} =
25        $module;
    $mp->{$name}{instance}{$instance_name}{library} =
        $library;

    my $ref = $mp->{$name}{instance}{$instance_name};

    #####
    # split up port map and assign connections
    my $port_map = $rest;
    $port_map =~ s/^\.*?\bport\s+map\s*(\s*(.*)/$1/is or
35    &ribbit
        ("no port map for instantiation $instance_name
($command)");

    my ($pre,$pm,$end) = &HDL_Count_Parentheses
40    ($port_map);
    $pm =~ s/\s+//sg;

    my @connections = split (/\s/, $pm);
    foreach $c (@connections)
45    {
        my ($module_port,
            $external_port) =
            split (/\s=>/, $c);

50        #####
        # a big hack for now,
        # crush all those (0) cases which convert
        # std_logic_vector (0 downto 0) to std_logic
        $external_port =~ s/(0\)$//s;

55        $ref->{connection}{$module_port} = $external_port;
    }
}
}
60    return ($mp);
}

ribbit "language ($language) not understood\n";
```

```

}

sub HDL_List_Ports_For_Module
{
5   my $module_hash = shift or ribbit "no hash";
    my $module = shift or ribbit "no module";

    my $tmp = &HDL_Get_By_Path ($module_hash, ".$module.port_order");
10   my @module_port_array = @$tmp;

    my $list_ports_for_string;
    foreach $port (@module_port_array)
    {
15       $list_ports_for_string .=
           "$port |
           $module_hash->{$module}{signal}{$port}{width} |
           $module_hash->{$module}{signal}{$port}{direction},
           ";
    }
20   &List_Ports_For ($module, $list_ports_for_string);
}

#####
25 # HDL_Munge_Data
#
# For assigning items in which you don't know the declaration order,
# e.g for stuff that happens across module boundaries, you have to wait
# until all data is known before you can start processing it. Its
30 # also good to do as much work here as possible because you only have
# to do it once here rather than once for each of the supported
# languages.
#####

35 sub HDL_Munge_Data
{
    my $hash = shift or ribbit "no hash";

    foreach $module (&HDL_Get_Keys_By_Path($hash, "", "", 0))
40     {
        my $boo = $hash->{$module}{instance};
        my @asdf = keys (%$boo);
        foreach $instance (&HDL_Get_Keys_By_Path($hash, "$module.instance", 1))
        {
45             my @tmp = keys (%{$hash->{$module}{instance}});
            my $instantiated_module = &HDL_Get_By_Path
                ($hash,
                "$module.instance.$instance.module", "");

50             #####
            #assign instance parent name and hash
            $hash->{$module}{instance}{$instance}{hash} =
                $hash->{$instantiated_module} or warn
55             "module ($module) instantiates unknown module
            ($instantiated_module)\n";

            $hash->{$module}{instance}{$instance}{parent} =
                $hash->{$module};

60             #####
            # if you have a module declaration "<module> <instance> ((.a)
            # b)" and module.a is an output, then what you're really

```



```
# saying is (assign b = <instance>.a). We do that assignment
# now for all output ports of the $instance here.
```

```
my $mod_sig = &HDL_Get_By_Path
($hash,
"$module.instance.$instance.connection");

foreach $port (keys (%$mod_sig))
{
    if (&HDL_Get_By_Path
        ($hash, "$instantiated_module.signal.$port.direction", 1)
        =~ /^out/i)
    {
        $hash->{$module}{signal}{$$mod_sig{$port}}{assignment}
        = "$instance.$port";
    }
}
}
```

```
#####
# HDL_Get_Module_Info_From_File
#
# wrapper around get_module_info. Does language determination based
# upon file extension.
#####
```

```
sub HDL_Get_Module_Info_From_File
{
    my %h = @_;

    $h{file} or ribbit "no file specified\n";

    if (!$h{language}) # if language not specified, determine from file
                        # suffix. Default is verilog
    {
        $h{language} = "verilog";
        $h{language} = "vhdl"
            if ($h{file} =~ /\.\vhdl?/i);
    }

    my $module_string = &HDL_Read_File($h{file});

    my $module_hash = &HDL_Get_Module_Info ($module_string,
                                            $h{language},
                                            $h{hash});

    return ($module_hash);
}
```

```
#####
# HDL_Get_Module_Info_From_Files
#
# wrapper around get_module_info_from_file. Munges data after all
# modules are known.
#####
```

```
sub HDL_Get_Module_Info_From_Files
{
    my %h = @_;
    $h{file_array} or ribbit "no file array";

    my %hash;
```

09880106-061201
102190-90T0896

```

foreach $file (@{h{file_array}})
{
    my $hash = &HDL_Get_Module_Info_From_File(file => $file,
                                                hash => \%hash,
                                                language => $h{language});
}

```

```

#now we have all the data given to us from the file list.
#play with it a bit
&HDL_Munge_Data(\%hash);

```

```

return (\%hash);
}

```

```

#####
# HDL_Get_Keys_By_Path
#
# Sugar around Get_By_Path, it just assumes value gotten is a hash and
# returns its keys. Could do error checking with ref operator

```

```

sub HDL_Get_Keys_By_Path
{

```

```

    my $pHash = shift or ribbit "no hash";
    my $rel_path = shift;
    my $quiet = shift;
    my $debug = shift;

```

```

    warn "\n\nHDL_Get_Keys_By_Path:"
        if $debug;

```

```

    my $ref_hash = &HDL_Get_By_Path($pHash,$rel_path,$quiet,$debug);

```

```

    my @return_array = keys (%$ref_hash);
    warn "done with HDL_Get_Keys_By_Path returning (@return_array)\n"
        if $debug;
    return (@return_array);
}

```

```

#####
# HDL_Set_By_Path (NOT FINISHED)
#

```

```

# An experiment in setting by path. currently we use the all powerful
# arrow operator
sub HDL_Set_By_Path
{

```

```

    my $pHash = shift or ribbit "no hash";
    my $rel_path = shift;
    my $value = shift or ribbit "no value";
    my $make_new_path = shift;

```

```

    $rel_path =~ s/\s+//g;
    $rel_path =~ s/^\s*\.(.*?)\.(.*?)\s*/$1/; #take off initial and final .s

```

```

    $rel_path =~ s|\.|\)|\(|g; #change a.b to a){b
    #to be continued

```

```

    #$rel_path =~ ;
}

```

```

#####
# HDL_Get_By_Path
#

```

```

# You pass in hash and a "." separated path. It progresses down the
# hash tree and gives you your result. If it cannot go down the tree
# and $quiet is FALSE. It ribbits where it failed and displays the
# leaves available at that point. if ($quiet) it just returns ""

```

09880106-061201

#####

sub HDL_Get_By_Path

{

my \$pHash = shift or ribbit "no hash";
my \$rel_path = shift;
my \$quiet = shift;
my \$debug = shift;

\$rel_path =~ s/^\s*\.\?(\.*)\s*\$/\$1/s;

warn "\n\nHDL_Get_By_Path: rel path is \$rel_path\n"

if (\$debug);

my @path = split (/^\s*\.\s*/,\$rel_path);

my \$indent;

while (\$child = shift (@path))

{

my \$child_options = join (" or\n", (sort (keys %\$pHash)));

if (\$debug)

{

warn "\$indent\$pHash ->(\$child)\n";

\$indent .= " ";

}

\$pHash = \$pHash->{\$child};

if (!\$pHash)

{

return ("") if \$quiet;

&ribbit ("(\$child) unknown in \$rel_path\n\n"

."known options are:\n(\$child_options)\n");

}

}

if (\$debug)

{

warn "\$indent returning \$pHash\n";

}

return (\$pHash);

}

#####

HDL_Get_Module_By_Instance_Path

#

#You pass in hash and a "." separated path. It progresses down the

instantiation list and returns the module name at the end of the tree

#####

sub HDL_Get_Module_By_Instance_Path

{

my (%h) = @_;

\$h{path} or ribbit "no path specified";

\$h{hash} or ribbit "no hash specified";

\$h{path} =~ s/^\s+//g;

my @path = split (/^\./,\$h{path});

my \$stop = shift @path;

\$stop = shift @path

if (\$stop =~ /^^\s*\$/);

my \$hash = \$h{hash};

09020106 "061201

```

while (@path)
{
    my $leaf = shift @path;
    $stop = &HDL_Get_By_Path
5      ($hash, "$stop.instance.$leaf.module");
}
return ($stop);
}

10 #####
# HDL_Get_Parent_Connection
#
# returns the full path of the signal that connects to an instance
#####
15 sub HDL_Get_Parent_Connection
{
    my (%h) = @_;

20    $h{path_and_signal} or ribbit "no path and signal specified";
    $h{hash} or ribbit "no hash specified";

    my @path_a = split (/\\s*\\.\\s*/s, $h{path_and_signal});
    my $signal = pop (@path_a) or ribbit "we get no signal";
25    my $instance = pop (@path_a) or ribbit "we get no instance";

    my $path = join("\\.", @path_a);
    my $module =
30        &HDL_Get_Module_By_Instance_Path (path=> $path,
                                            hash=> $h{hash}
                                            );

    my $parent_connection =
        &HDL_Get_By_Path ($h{hash},
35        "$module.instance.$instance.connection.$signal");

    return "$path\\.$parent_connection";
}
1;

```

09880106 061201 102190 90100000

Get_Sopc_Path.pl

```
#####  
5 # get_sopc_path  
#  
# Figure out, based on the system environment, where we should  
# look for components. The list of directories is delimited by  
# plus (+) symbols, does not include the implicit '.' and socp  
10 # components directories. STDOUT will receive the list.  
#  
#####  
  
$| = 1;          # set flushing on STDOUT  
  
15 # NOTE: some perl expert should change this to call getenv() or whatever.  
print $ENV{SOPC_BUILDER_PATH};
```

09880106-061201
T02T50" 90T08860

Mifsim1.pl

sub Emit_Bin_Data

{
my \$data_string;
(\$data_string) = (@_);

my \$data_val = eval ("0x\$data_string");

my \$bit = 0;
my \$result = "";
for (\$bit = 0; \$bit < \$DATA_WIDTH; \$bit++)

{
if (\$data_val % 2 == 0) {
\$result = "0" . \$result;
} else {
\$result = "1" . \$result;
}

\$data_val = int (\$data_val / 2);
}

print "\$result\n";

\$DATA_WIDTH = shift (@ARGV);

while (<>)

{
if (/^--/)
{
It's a MIF "--" comment at the start of a line:
Keep comments, because I love them.
s/^--/\\/\\/;/;
print;
next;
}

s/\\s*//g; # Strip all whitespace, because I hate whitespace.

if (/((\d+)\.(\w+));/)
{

my \$addr = \$1;
my \$data = \$2;
printf ("%X\n", \$addr);
print "\$data\n" if !\$DATA_WIDTH;
&Emit_Bin_Data(\$data) if \$DATA_WIDTH;
next;

next;

}

09880106 "061201
T02T00" 9088860

MK_Custom_SDK.pl

#!/bin/sh
exec perl - "\$@" <<\ENDOFPERL

#!/perl

require Strict;
use mk_custom_sdk;

2000 August
dvb \ Altera Santa Cruz

see if any of the arguments is "--help",
and show some help if so. Otherwise, call main.
#

{
my \$i;

for(\$i = 0; \$i < scalar(@ARGV); \$i++)
{
if(\$ARGV[\$i] eq "--help")
{
usage();
exit 0;
}
}
}

mk_custom_sdk(@ARGV);

end of file

09880106 "061201

MK_Systembus.pl

#####

mk_systembus.pl

#

This Perl-script is the "business end" of the
Nios System Bus Wizard. The Wizard itself is a GUI-layer
which quizzes the user and passes his(her) choices
along to this very script.

#

The kind of user socket we build depends on the
parameters we get. The parameters are "named arguments,"
Named arguments are one long comma-delimited string,
a list of 'normal' command-line arguments, or any combination
of both (we just smash all the command-line arguments together
into one long string anyhow).

#

The comma-delimited elements have the form:

<arg_name> = <value>.

#

For a list of all the argument-names and their allowed values,
see the table below.

#

use wiz_utils;

use mk_custom_sdk;

use pbm_gen;

use crush_names;

#use v2vhd;

#use strict;

#use Win32;

#use Win32::Process;

#####

Mk_SystemBus

#

Executes all the functions of the System Bus MegaWizard.
All the peripherals (and nios-cores) that this uses must have
already been built by the other "Mk" functions.

#

Because this function takes listref and hashref arguments,
it doesn't use PARSE_NAMED_ARGS.

#

This function takes, as its arguments:

#

\$Mk_SystemBus_Doc=<<END_OF_DOCUMENTATION_STRING ;

LONG NAME SHORT NAME DEFAULT DESCRIPTION

mainmem_module	--none--	--none--	SDK will target programs here.
skip_synth	--none--	0	*boolean* do synth or not?
hdl_language	hdl	verilog	*(verilog vhd ahdl)* for wrapper.
device_family	--none--	APEX20KE	target device (chip) family.
compiler	--none--	quartus	*(max\ plus2 quartus)* P&R tool
clock_freq	clk_f	33333300	Input clock rate in Hz.
do_build_sim	sim	1	*boolean* do make system sim?
do_optimize	optimize	1	*boolean* assume optimized HDL?
leo_flatten	flatten	1	*boolean* Leo's hier-flatten option
leo_area	area	0	*boolean* Leo optimize for area.

Principal_Tri_State_Data_Bus --none-- --none-- Direct-to-CPU Fast I/O

These arguments are used to build the custom SDK. We have to tolerate
them here, even though we don't use them

mainmem_module --none-- --none-- Where to put programs.

09880106 "061201

datamem_module	--none--	--none--	Where to put variables/stack.
gdbcomm_module	--none--	--none--	Debug on this uart.
maincomm_module	--none--	--none--	Yak on this uart.
germs_monitor_id	--none--	--none--	Print this at boot-time.

5

END_OF_DOCUMENTATION_STRING

#

#####

sub Mk_SystemBus

10

```
{
    my ($arg, $user_defined, $db_Sys, $db_PTF_File)
        = &Process_Wizard_Script_Arguments ($Mk_SystemBus_Doc, @_);
```

15

```
    my $sys_name = $$arg{system_name};          # too handy to pass
                                                # up.
```

```
    if ($$arg{compiler} eq "max+plus2")
```

```
    {
        if ($sys_name =~ /\^(\\w{32})\\w+/)
```

20

```
        {
            die "Sorry, max+plus2 will not accept a system name\n".
                " that is greater than 32 characters. Perhaps you\n".
                " should use ($1) as your system-name instead.\n";
        }
    }
}
```

25

#####

```
# First, but by no means most, build a custom SDK
# for all that annoying hardware we're about to build:
#
```

30

```
&mk_custom_sdk( "--sopc_directory=$$arg{sopc_directory}",
                "--system_directory=$$arg{system_directory}",
                "--system_name=$$arg{name}",
                "--sopc_lib_path=$$arg{sopc_lib_path}"
                );
```

35

```
$do_optimize = $$arg{do_optimize};
```

```
if ($do_optimize) {
```

```
    $suffix = ".txt";
```

```
    unlink ("ABRAHAM LINCOLN_NO_OPTIMIZE");
```

```
} else {
```

```
    open (ABRAHAM LINCOLN_NO_OPTIMIZE, ">ABRAHAM LINCOLN_NO_OPTIMIZE");
    close ABRAHAM LINCOLN_NO_OPTIMIZE;
```

```
}
```

45

```
&Progress ("Starting generation for system: $sys_name.");
```

```
# Get "{quartus,leonardo,modelsim}_define.v, and all that stuff:
```

```
#
```

50

```
&Copy_Tool_Control_Files ($arg);
```

#####

```
# Run Client Generator-Programs
```

```
#
```

55

```
# Loop through every system component (module) specified in the PTF-file
# and run the corresponding generator-script.
```

```
#
```

```
my @module_name_list = &Run_Generator_Programs ($arg, $db_Sys);
```

60

#####

```
# Update PTF database.
```

```
#
```

```
# The PTF database we read-in earlier is now
```

T02190" 90T08860

```

# out-of-date. We just ran a bunch of "generator programs."
# One consequence: The PTF-file has been thoroughly modified.
# Read it in again:
#
5 my ($arg, $user_defined, $db_Sys)
    = &Process_Wizard_Script_Arguments ($Mk_SystemBus_Doc, @_);

#####
# Build the system-module and PBM:
10 #
# Internally, this function generates an elaborate database hash.
# It returns a reference, so that we can extract a few key
# elements later (name of core-module and such).
my @sys_synth_file_list = ();
15 &Progress ("Making PBM (bus) and system (top) modules.");
my $pSys = &Generate_PBM_And_System ($arg, $db_Sys);

my $synth_list_ref = $$pSys{synth_file_list};
push (@sys_synth_file_list, @$synth_list_ref);
20 #####
# Synthesis-Files list
#
my @synthesis_file_list = &Get_Synthesis_File_List($pSys, $db_Sys,
25                                     \@sys_synth_file_list,
                                     @module_name_list
                                     );

30 # The name of the synthesizable core-file itself may have been changed.
$arg->{core_name} = $pSys->{core_name};

# The Java wizards need to see this string to know everything is OK.
# At this point, we consider everything "OK"-enough to
35 # allow the Java wizards to create a subsequently-editable
# "custom megafunction variation"
#
&Progress ("VPP HDL-GENERATION SUCCESSFUL");

40 # Write-out a list of the HDL files, in case the user
# wants to synthesize the design his/herself:
#
&Create_HDL_File_List_File ($$arg{system_directory}, $sys_name,
45                                     @synthesis_file_list);

# Create simulation project, if so ordered:
#
50 &Create_Sim_Project ($arg, @synthesis_file_list) if $$arg{do_build_sim};

#####
# Time to Synthesize!
#
# (Leonardo, if you please...)
55 #
# Pass along our $args-hash, so that Leo can know everything we know.
#
&Run_Leonardo ($arg, @synthesis_file_list);
60 }

#####
# Run_Leonardo

```

09880105 061201
101219Z

```

#
# We run Leonardo Spectrum (aka "spectrum") from the
# command-line by pointing it to a command-file. The
# command-file contains, mostly, a list of all the HDL-files
5 # we want to synthesize, plus a few simple settings.
#
# This isn't really a "subroutine" in the conventional sense,
# because it only gets called from one place. It's just
# for tidy code-partitioning.
10 #
#####

# I got these keys (family-names) from some file in the quartus/bin
# directory called "package.dat".
15 #
# I did it again in the maxplus2/ directory to get the reset.
#
# I got these values by typing "ls *.syn" in Leo's "lib"-directory:
#
20 my %Leonardo_Device_Family_Decoder_Ring = (
    APEX20K      => "apex20",
    APEX20KE     => "apex20e",
    APEX20KC     => "apex20c",
    EXCALIBUR_ARM => "excalibur_arm",
25 EXCALIBUR_MIPS => "excalibur_mips",
    MERCURY      => "mercury",
    ACEX1K       => "acex1",
    FLEX10K      => "flex10",
    FLEX10KA     => "flex10a",
30 FLEX10KB      => "flex10b",
    FLEX10KE     => "flex10e",
);
sub Run_Leonardo
{
35   my($arg, @unsorted_input_file_list) = (@_);

   #@input_file_list = sort @unsorted_input_file_list;
   # the last entry in the input_file_list is always the top-level module.
   # we want the top-level module to appear first on the list.
40   @input_file_list = @unsorted_input_file_list;
   my $top_level_module = pop @input_file_list;
   unshift @input_file_list, $top_level_module;

45   my $sys_name = $$arg(system_name);    # too handy to pass up.

   my $freq_in_MHz = $$arg(clock_freq)/1000000;    # note, need not be int.
   my $tcl_file    =
       "$$arg(system_directory)/$sys_name\_leonardo\_tcl\_script.tcl";
50   my $command_file =
       "$$arg(system_directory)/$sys_name\_leonardo\_commands.cmd";

   my $target = $$arg(device_family);
       $target = $Leonardo_Device_Family_Decoder_Ring{$target};
55   die "ERROR Run_Leonardo NULL TARGET! ($$arg(device_family)) does not map!\n"
       if ($target eq "");

60   my $hierarchy_option = $$arg(leo_flatten) ? "hierarchy_flatten" :
                                           "hierarchy_preserve" ;
   my $optimize_option = $$arg(leo_area)    ? "area" :
                                           "delay" ;

```

```

my $LEO_CMD_FILE_TAIL=<<END_OF_TAIL;
-product=ls1
-target=$target
5  -macro
  -$optimize_option
  -max_frequency=$freq_in_MHz
  -effort standard
  -$hierarchy_option
10 -pass={1}
END_OF_TAIL

  open (LEO_CMD, "> $command_file") or die "
    ERROR: couldn't open $command_file: $!";
15 print LEO_CMD "leonardo_define.v \n";
  foreach $file (@input_file_list)
    { print LEO_CMD "$file \n"; }          # List input files.
  print LEO_CMD "$$arg{core_name}.edf \n"; # Next, output file.
  print LEO_CMD "-module=$$arg{core_name} \n"; # Top module.
20 print LEO_CMD $LEO_CMD_FILE_TAIL;
  if ($do_optimize) {
    print LEO_CMD "-file=$tcl_file \n";
  }
  close (LEO_CMD);
25
# at this moment, we don't do a tcl file unless we do_ip.
  if ($do_optimize) {
    open (LEO_TCL, "> $tcl_file") or die "
      ERROR: couldn't open $tcl_file: $!";
30 print LEO_TCL "\n"; # just enough exist, if nothing else written to it.
    print LEO_TCL "do_ip -target $target ";
    print LEO_TCL "-area " if $$arg{leo_area};
    print LEO_TCL "-delay " if !$$arg{leo_area};
    print LEO_TCL "-effort standard ";
35 print LEO_TCL "-design $$arg{core_name} -pass={1} ";
    print LEO_TCL "-hierarchy flatten " if $$arg{leo_flatten};
    print LEO_TCL "-hierarchy preserve " if !$$arg{leo_flatten};
    print LEO_TCL "-output $$arg{core_name}.edf ";
    print LEO_TCL "{ leonardo_define.v ";
40 foreach $file (@input_file_list)
    { print LEO_TCL " $file"; }          # List input files.
    print LEO_TCL "}" \n";
    close (LEO_TCL);
  }
45
# The following section of code is defunct. The problem is, we don't have a
# full license to Leonardo. If we did, then using a full tcl script would be
# a really really good idea. But we don't. We have a stupid level 1 license.
# This level 1 license will allow us to use the "do_ip" function in a tcl
50 # script like above (thank goodness), but it won't allow us to run the entire
# Leo project from a tcl script. With hope that we will eventually be able to
# run Leonardo through a tcl script, I will keep this around, albeit
# commented, as a dedication to untapped powers of tcl-enabled synthesis.
#
55 # $input_file_list_string = join '\+', @input_file_list;
# &Vpp (
#   split (/s+/, "-Q -R -X tcl"),
#   "-D", "$$arg{system_directory}",
#   "-P", "$$arg{system_name} . "_",
60 #   "SYSTEM_NAME" = $$arg{system_name}",
#   "TOP_MODULE_NAME" = $$arg{core_name}",
#   "INPUT_FILE_LIST_STRING" = $input_file_list_string",
#   "SYSTEM_CLOCK_FREQUENCY" = $$arg{clock_freq}",

```

```

#         "DEVICE_FAMILY"                = $$arg(device_family)",
#         "DO_OPTIMIZE"                  = $$arg(do_optimize)",
#         "$$arg(sopc_directory)/bin/project_synthesis_script.vpp",
#     );

```

```

5   my $spectrum_bin_dir = "$$arg(sopc_directory)/";
    $spectrum_bin_dir .= "bin/spectrum/bin";
    $spectrum_bin_dir .= "/win32" if ($^O =~ /(MSWin|cygwin)/i);
    my $spectrum_command = "$spectrum_bin_dir/spectrum";
10  my $spectrum_command_line = $spectrum_command
    . " -command_file="
    . $command_file;

```

```

    if ($$arg(skip_synth))

```

```

15  {
    print STDERR "
        Nios system module $sys_name *not synthesized*
        You must synthesize this module before you can place
        and route in Quartus.\n";

```

```

20  } else {
    &Progress ("Launching synthesis tool.");

```

```

    open (ABRAHAM LINCOLN_STEALTH, "");
    close ABRAHAM LINCOLN_STEALTH;
25  my $error_code = &System_Win98_Safe ($spectrum_command_line);
    open (ABRAHAM LINCOLN_NO_STEALTH, "");
    close ABRAHAM LINCOLN_NO_STEALTH;

```

```

    if ($error_code == 2) {
30      die "
        Leonardo Spectrum was unable to run due to a bad
        or nonexistant license file.
        Be sure that you have a valid license to run
        the \"Altera OEM\" version of Leonardo Spectrum.
35      You can obtain a license from \"www.altera.com\".\n";
    }

```

```

    if ($error_code == 1) {
40      die "
        Leonardo Spectrum did *not* run successfully.
        Spectrum has reported an error in a design file.

```

```

        You must resynthesize this module before you can place
        and route in Quartus.\n";
    }

```

```

45  if ($error_code != 0) {
    die "
        Leonardo Spectrum did *not* run successfully.
        Spectrum quit because of an unknown error: $error_code.

```

```

50  You must resynthesize this module before you can place
    and route in Quartus.\n";
    }

```

```

    &Progress ("Spectrum Done.");

```

```

55  if ($arg->{compiler} =~ /max\+plus2/i) {
    print STDERR"
        Be sure that your Interfaces-->EDIF Netlist_Reader Settings
        specify \"Exemplar\".\n";
60  }

```

```

}

```

09380106 051201

}

```
#####  
# Is_Leonardo_Licensed_for_Verilog  
#  
# Tests whether Leonardo Spectrum has access to a license  
# that allows it to process verilog. It does this by  
# producing a very simple verilog file, and seeing if  
# spectrum can synthesize it. If it fails on the license,  
# then it must not have a license for verilog.  
#  
#####
```

```
sub Is_Leonardo_Licensed_for_Verilog
```

```
{  
    my ($arg) = (@_);  
    my $is_licensed = 0;  
    my $t = "test_for_leonardo_verilog_license_file";  
  
    &Progress ("Testing synthesis tool license.");  
    # we need a blank verilog file just to run on.  
    open (LEO_DEF, ">$t.v");  
    print LEO_DEF ("module $t (i, o); input i; output o; assign o=i;  
endmodule\n");  
    close LEO_DEF;  
  
    my $spectrum_bin_dir = "$arg{sopc_directory}"/;  
    $spectrum_bin_dir .= "bin/spectrum/bin";  
    $spectrum_bin_dir .= "/win32" if ($^O =~ /MSWin/i);  
    my $spectrum_command = "$spectrum_bin_dir/spectrum";  
    my $spectrum_command_line = $spectrum_command  
        . " -product=ls1"  
        . " -target=apex20"  
        . " $t.v"  
        . " $t.edf"  
        . " > $t.output" ;  
    open (ABRAHAM LINCOLN STEALTH, "");  
    close ABRAHAM LINCOLN STEALTH;  
    my $error_code = &System_Win98_Safe ($spectrum_command_line);  
    open (ABRAHAM LINCOLN_NO_STEALTH, "");  
    close ABRAHAM LINCOLN_NO_STEALTH;  
  
    if ($error_code == 1) {  
        $is_licensed = 0;  
        print STDERR ("Unable to determine find verilog license for Leonardo.  
        Using unlicensed synthesis path.\n");  
    } elsif ($error_code == 0) {  
        $is_licensed = 1;  
    } else {  
        print STDERR ("Unable to determine licensing for Leonardo.  
        Using unlicensed synthesis path.\n");  
        $is_licensed = 0;  
    }  
    # clean up directory  
    unlink ("$t.v", "$t.log", "$t.sum", "$t.tcl", "$t.xdb", "$t.xrt",  
        "$t.edf", "$t.xrf", "$t.output");  
    return ($is_licensed);  
}
```

```
#####  
# Run_Generator_Programs  
#  
# Given (a reference to) the PTF "SYSTEM" section and (a reference to)
```

09880106 "051201"

```

# the %arg-hash, we have enough information to run through all the
# system's sub-modules and run their respective generator-programs
# (as-listed in their "class.ptf" file), if any.
#
5 # For modules that don't explicitly list a generator-program, we run
# the "default_generator_program," on their behalf. Other modules may
# specifically request that no generator program be run at all.
#
# This function returns a list of all "enabled" modules in the system.
10 # This list includes the master.
#
#####
sub Run_Generator_Programs
{
15   my ($arg, $db_Sys) = (@_);

   my @module_name_list = ();
   my $num_children     = get_child_count($db_Sys);
   for (my $child_index = 0; $child_index < $num_children; $child_index++) {
20       my $db_Module = &get_child ($db_Sys, $child_index);
       next unless get_name ($db_Module) eq "MODULE";

       # Don't waste our time on disabled modules.
25       next if !&PTF_Get_Boolean_Data_By_Path ($db_Module,
                                                "SYSTEM_BUILDER_INFO/Is_Enabled");

       my $mod_name = &get_data ($db_Module);
       push (@module_name_list, $mod_name);    # Good to know later.
30       # Open-up this module's "class.ptf" file.
       # The "class" name of this peripheral is, by definition,
       # the same name as the "components/" directory in which
       # its class.ptf file resides.
35       #
       my $module_class = &PTF_Get_Required_Data_By_Path ($db_Module, "class",
                                                         "No class specified for module: $mod_name");

       my $module_lib_dir =
40         &Find_SOPC_Component_Directory ($module_class, $$arg{sopc_lib_path});

       my $db_Class_File =
         &PTF_New_Required_Ptf_From_File (" $module_lib_dir/class.ptf",
45         "No 'class.ptf' file found for module $mod_name");

       my $db_Module_Class =
         &PTF_Get_Required_Child_By_Path ($db_Class_File, "CLASS",
50         "Bad or corrupt 'class.ptf' file for module $mod_name");

       my $lib_generator_program = &get_data_by_path ($db_Module_Class,
                                                      "ASSOCIATED_FILES/Generator_Program");

       # If the library component didn't specify a generator program,
       # then give it the generic (default) one:
55       #
       my $generator_program = "$module_lib_dir/$lib_generator_program";
       $generator_program =
         "$$arg{sopc_directory}/bin/default_generator_program.pl"
60         if (($lib_generator_program eq "" ) ||
            ($lib_generator_program =~ /^--default--$/i) );

       if (($lib_generator_program !~ /^--none--$/i)) {

```

09880106 "061201

```

# for now, complain bitterly if the generator program is not
# apparently a Perl-script:
#
$generator_program =~ /\.pl$/ or die "
5   Illegal Generator program '$generator_program' for $module_class:
    Generator programs must be perl-scripts.\n";

my
$generator_cmd = "$$arg{sopc_directory}/bin/iperl ";
$generator_cmd .= "-I$$arg{sopc_directory}/bin ";
10  $generator_cmd .= "$generator_program ";
$generator_cmd .= "      --system_name=$$arg{system_name}      ";
$generator_cmd .= "      --target_module_name=$mod_name      ";
$generator_cmd .= "      --system_directory=$$arg{system_directory} ";
$generator_cmd .= "      --sopc_directory=$$arg{sopc_directory} ";
15  $generator_cmd .= "      --sopc_lib_path=$$arg{sopc_lib_path} ";
$generator_cmd .= "      --generate=1 ";
$generator_cmd .= "      --verbose=$$arg{verbose} ";

open (ABRAHAM_LINCOLN_STEALTH, "");
20  close ABRAHAM_LINCOLN_STEALTH;
my $error_code = &System_Win98_Safe ($generator_cmd);
open (ABRAHAM_LINCOLN_NO_STEALTH, "");
close ABRAHAM_LINCOLN_NO_STEALTH;
$error_code == 0 or die "
25  Error: Generator program '$generator_program'
    for module $mod_name did NOT run successfully.\n";
}
}

30  return @module_name_list;
}

#####
# Get_Synthesis_File_List
#
35  # Gets an array of files to be synthesized and massages them for
# Max+Plus2 if required.
#
# If Max+2 is p+r tool, we convert MIF files and other files
40  # to max+2 friendly files.
#####

sub Get_Synthesis_File_List
{
45  my ($pSys,
      $db_Sys,
      $pAdditional_Synth_Files,
      @module_name_list,
      ) = @_;

50  my @synthesis_file_list;
my @mif_file_list;
my @additional_file_list;
foreach $module_name (@module_name_list)
55  {
    my $db_Module =
        &PTF_Get_Required_Child_By_Path ($db_Sys,
            "MODULE $module_name",
            "I could have sworn $module_name was in here somewhere!");

60  my $hdl_file_data =
        &get_data_by_path ($db_Module, "HDL_INFO/Synthesis_HDL_Files");
    push (@synthesis_file_list, split(/\s*\s*/, $hdl_file_data));
}

```

09880106 "061201


```

my $mif_file_data =
    &get_data_by_path ($db_Module, "HDL_INFO/MIF_Files");
push (@mif_file_list, split (/\\s*\\,\\s*/,$mif_file_data));

5
    my $additional_file_data =
        &get_data_by_path ($db_Module,
"HDL_INFO/Other_Files_Subject_To_MPII_Length_Limit");
    push (@additional_file_list, split (/\\s*\\,\\s*/,$additional_file_data));
10
}

push (@synthesis_file_list, @$pAdditional_Synth_Files);

15
#####
# Orion is sticking an option for vhd1 translation back here
# because of the synthesis-for-one-HDL fiasco. Ideally, Leonardo will
# be fixed before we ship this. If, however, it isn't, here is our
# backup plan.
20
#
if (0) #set to 1 if you want vhd1 translation on vhd1 files
{
    # $pSys->{hdl_language} really should be $pSys->{leonardo_language}
    if ($pSys->{hdl_language} =~ /^vhd1/i)
    {
25
        @synthesis_file_list =
            &V2VHD_Files(
                "\\define LEONARDO_SPECTRUM",
                ".",
30
                @synthesis_file_list
            );
    }
}
#
35
#####

#####
40
# Max+Plus 2 has "issues" with quartus problems with large names.
# It doesn't like names that are bigger than 32 characters.
# and it likes mif files a certain way. We oblige here.

if ($pSys->{compiler} eq "max+plus2") {
45
    &Progress ("Imposing 32-char name limit for MaxPlus+II.");

    my $length_limit = 32;
    my %Conversion_Hash;      # stores names that have been converted
    my %Converted_Filenames;  # stores filenames that have been converted
50
    my $sys_name = $pSys->{system_name};    # too handy to pass up.

    $Conversion_Hash{$sys_name} = $sys_name; #keeps $sys_name the same
    $pSys->{core_name} = &Crush_Line ($pSys->{core_name},
55
                                    $length_limit,
                                    \\%Conversion_Hash);

    @synthesis_file_list =
        &Crush_Names_That_Are_Bigger_Than_Max_Width_Characters
60
        (
            \\%Converted_Filenames,
            \\%Conversion_Hash,
            $length_limit,
            @synthesis_file_list

```

09380106 "051201

```

    );

    my @wrapper_files = ($pSys->{wrapper_file});
    push (@wrapper_files, $pSys->{inc_file})
5      unless $$pSys{inc_file} eq "";

    @wrapper_files =
      &Crush_Names_That_Are_Bigger_Than_Max_Width_Characters
10      (
        \%Converted_Filenames,
        \%Conversion_Hash,
        $length_limit,
        @wrapper_files
      );

15      &Make_MIF_Files_Max_Friendly (
        \%Converted_Filenames,
        \%Conversion_Hash,
        $length_limit,
20      @mif_file_list
      );

    }

    return (@synthesis_file_list);
25 }
#####
# Create_Sim_Project
#
# Builds a simulation-project (targeted at ModelSim) which will
30 # contain a simulatable version of the system-module.
#
#####
sub Create_Sim_Project
{
35   my ($arg, @hdl_file_list) = (@_);

   &Debug (0, "Creating simulation project directory for $$arg(system_name)");

   # we want to include absolute-path-name HDL files exactly
40   # as they appear, but relative path names have to
   # be re-referenced to "relative one-level-up."
   #
   my @include_list = ();
   foreach $include_file (@hdl_file_list) {
45     $include_file =~ tr|\||\./|;

     $include_file =~ s|^\.\/|\.\.\/|;
     my $is_absolute = $include_file =~ /\:/ ||
                       $include_file =~ /\^\.\/ ||
50     $include_file =~ /\^\/|/ ;

     $include_file = "../$include_file" if !$is_absolute;
     $include_file =~ s/\.v/\.v.txt/ if $$arg(do_optimize);

55     push (@include_list, $include_file);
   }

   #####
   # Emit the test-bench file:
60   #
   my $test_bench_name = "$$arg(system_name)_test_bench";
   my $test_bench_file = "$$arg(system_sim_dir)/$test_bench_name.v";
   &Debug (0, " Test bench file is: $test_bench_file");

```

09880106.061201

```

open (TESTOUT, "> $test_bench_file") or die "Couldn't open $test_bench_file";
my $old_out = select (TESTOUT);

print "
5   `timescale 1ns / 100ps
   `include \"modelsim_define.v\"
   ";
foreach $include_file(@include_list) {
10  print "\n`include \"$include_file\"";
}
print "
   `include \"test_equipment.v\"

   module $test_bench_name;
15     wire clk;
     wire reset_n;
     Clk_Gen Clk_Gen_33MHz (.clk (clk));
     Reset_Gen Reset_Gen_33MHz (.reset_n (reset_n));
   ";
20  &Declare_Wires_For_Connection_To ("$$arg{system_name}_core",
                                     "", "clk", "reset_n");
  &Instantiate_And_Connect ("$$arg{system_name}_core", ".");

25  print "\n endmodule \n";

  close (TESTOUT);
  select ($old_out);

30  ##### I WAS HERE #####
  # Put code to copy test equipment, modelsim.v, and mpf-file
  # also, I must modify mk_rom.pl to re-convert its MIF-file,
  # and the RAM verilog to have a behavioral model built-in.
  #
35  }

#####
# Create_HDL_File_List_File
40  #
# The user might want to synthesize all the files in the system-module
# himself. If so, it would be very useful to have a -list- of all
# the HDL-files in the project. This function here creates a list
# of all the HDL-files in the project. What a happy circumstance.
45  #
#####
sub Create_HDL_File_List_File
{
50   my ($system_dir, $sys_name, @synthesis_file_list) = (@_);

   my $hdl_list_file_name = $sys_name .
                           "_list_of_hdl_files_for_synthesis.txt";

55   my $full_hdl_list_file_path = "$system_dir/$hdl_list_file_name";

   my $hdl_list_file_header = <<EOM;
#####
#
60  # $hdl_list_file_name
#
# Automatically-created by Altera Excalibur Nios(TM) MegaWizard
#

```

```

# This file contains a list of all HDL files necessary
# synthesize the Nios system module named:
#
#           $sys_name
5 #
# HDL-file list follows:
#
EOM

10     open (HDL_LIST, "> $full_hdl_list_file_path") or die "
        ERROR: Couldn't open $full_hdl_list_file_path: $!";

        print HDL_LIST $hdl_list_file_header;

15     foreach $file (@synthesis_file_list)
        { print HDL_LIST "#     $file\n"; }

        close (HDL_LIST);

20     print STDERR "
        A list of HDL files necessary to synthesize
        the module $sys_name has been written to
        this file:

25         $hdl_list_file_name\n\n";
    }

#####
# System_Win98_Safe
30 #
# Win-98-safe "wrapper" for Perl's built-in 'system' command.
#
# Windows-98 can't handle an executable-name ($ARGV[0]) which
# has forward slashes in it. WinNT and Win2000 can handle
35 # either forward- or backward-slashes. So, if we notice that
# the operating system is Windows (or Cygwin), then we convert
# forward-slashes to backslashes in -only- the program-name
# part of the system-command. We leave all the arguments
# alone-- whether '/' or '\' is OK in an argument is entirely
40 # up to the program.
#
#####
sub System_Win98_Safe
{
45     my (@command_parts) = (@_);
    my $sys_cmd          = join (" ", @command_parts);

    $sys_cmd =~ /^\\s*(\\S+)\\s+(.*)$/ or die
        "System_Win98_Safe: Suspicious system-command: $sys_cmd";

50     my $program_path = $1;
    my $arguments      = $2;

    $program_path =~ s|/|\\|g if ($^O =~ /(MSWin|cygwin)/i);

55     my $new_sys_cmd = "$program_path $arguments";
    system ($new_sys_cmd);

    my $error_code = ($? >> 8);
60     return $error_code;
}

#####

```

09880106-061201

Execution begins here

#####

&Mk_SystemBus (@ARGV);

09880106-061201
T02T90" 90T09880

Nios-Convert.pl

```
#!/bin/sh
exec perl - "$@" <<\ENDOFPERL
5  #!perl

# We now use "nios-convert" in the HDK.
# in the HDK, we can't rely on any standard Perl libraries.
# ... alas, not even "Strict":
10 #use Strict;

# -----
# nios-convert

15 # -----
# ceil(x)
#
# standard math ceil
#
20 sub ceil
{
    $x = shift;

    return int($x) if ($x == int($x));
25     return int($x + 1);
}

# -----
# addTo(stringRef, list)
30 sub addTo
{
    my $stringRef = shift;

35     my $i;

    for($i = 0; $i <= $#_; $i++)
    {
        $$stringRef .= $_[ $i ];
40     }

    $$stringRef .= "\n";
}

# -----
45 # dprint(list...)
#
# print only if gDebug

my $gDebug = 0;

50 sub dprint
{
    my $i;

55     return if ($gDebug == 0);

    for($i = 0; $i < 30; $i++)
    {
        print shift;
60     }

    print "\n";
}
```

09330106-061201

```

# -----
# dateTime()
#
# returns a relatively nice date & time string
5 #
sub dateTime
{
    my ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdet) =
localtime(time);
10     $mon++;
        $year += 1900;

    my $d = sprintf("%04d.%02d.%02d",$year,$mon,$mday);
    my $t = sprintf("%02d:%02d:%02d",$hour,$min,$sec);

15     return "$d $t";
}

20 # -----
# readFile(fileName)
#
# returns the complete file contents
#
25 sub readFile
{
    my $fileName = shift;
    my $bunch;
    my $result;
30     my $did;

    if(open(FILE,$fileName))
    {
        binmode FILE;
        while(read(FILE,$bunch,32000))
        {
            $result .= $bunch;
        }
        close FILE;
40     }

    return $result;
}

45 # -----
# writeFile(fileName,contents)
#
# creates new file and writes entire
50 # file contents. Return "ok" if so,
# or "" if not.
#
sub writeFile
{
55     my $fileName = shift;
        my $contents = shift;
        my $did;

    #
60     # Delete existing file, if any.
    #
    unlink ($fileName) if(-e $fileName);

```

```

    $ddid = open(FILE, ">$fileName");
    if($ddid)
    {
        binmode FILE;
        print FILE $contents;
        close FILE;
        return "ok";
    }

    return "";
}

# -----
# srec2hash(giantFileString)
#
# Given a giant string containing
# one entire srec file, return an
# associative array where each entry
# has a byte-address key, and its
# contents.
#
# This will of course be moderately
# gigantic, but who's counting?
#
# We just presume that RAM is cheap
# and plentiful and works, too.
#
sub srec2hash
{
    my $srecString = shift;
    my %hash;

    my $srecord;
    my $recordType;
    my $recordLength;
    my $recordChecksum;
    my $recordAddress;
    my $recordData;
    my $addressStringLength;

    foreach $srecord (split("\n", $srecString))
    {
        $srecord =~ s/\r//g;    # kill ^M's
        dprint "record $srecord";
        if($srecord =~ /^S([123])(..)(.*)($/))    # an S record we
        can use
        {
            $recordType = $1;
            $recordLength = hex($2)-1;
            $a = $3;
            $recordChecksum = $4;

            $recordLength -= $recordType + 2;
            $addressStringLength = ($recordType + 1) * 2;
            $recordAddress = hex(substr($a, 0, $addressStringLength));
            $recordData = substr($a, $addressStringLength);

            while(length($recordData))
            {
                $hash{$recordAddress} = hex(substr($recordData, 0, 2));
                $recordData = substr($recordData, 2);
            }
        }
    }
}

```

09880106 "061201


```

                                $recordAddress++;
                                }
                        }
5   dprint "srec hash keys";
    foreach $i (sort(keys(%hash)))
    {
        dprint"$i:$hash{$i}";
    }
10   return %hash;
    }

15   sub mifRadixFromText
    {
        my $mifRadix = shift;

20       return 10 if($mifRadix eq "UNS"
                    or $mifRadix eq "DEC");
        return 16;
    }

25   sub mifValueByRadix
    {
        my $mifData = shift;
        my $mifRadix = shift;

30       return hex($mifData) if($mifRadix == 16);
        return 1.0 * $mifData;
    }

    # -----
35   # mif2hash(giantFileString)
    #
    # Given a giant string containing
    # one entire mif file, return an
    # associative array where each entry
40   # has a byte-address key, and its
    # contents.
    #
    # This will of course be moderately
    # gigantic, but who's counting?
45   #
    # We just presume that RAM is cheap
    # and plentiful and works, too.
    #
    sub mif2hash
50     {
        my $srecString = shift;
        my %hash;

        my $mifrecord;
        my $mifWidth;
        my $mifAddressRadix;
        my $mifDataRadix;
        my $mifBytesPerData;
        my $mifData;
60     my $mifAddress;
        my $i;

        $mifWidth = 8;

```

09880106-061201

```

        foreach $mifrecord (split("\n",$srecString))
        {
            $mifrecord =~ s/\r//g; # kill ^M's
5      dprint "mif record $mifrecord";

        # Recognize 4 kinds of lines:
        #   WIDTH=x;
        #   ADDRESS_RADIX=HEX/DEC/UNS;
10     #   DATA_RADIX=HEX/DEC/UNS;
        #   addr:data;
        # Ignore anything else. We don't even care about the "DEPTH".
        #
            $mifrecord =~ s/[ \t ]//g; # kill white space
15     if($mifrecord =~ /^(.*)\=(.*)"$/ )
        {
            dprint "mif -- $1 = $2";
            if($1 eq "WIDTH")
            {
20                 $mifWidth = $2;
                $mifBytesPerData = int($mifWidth / 8);
                dprint "mif -- mifBytesPerData = $mifBytesPerData";
            }
            elsif($1 eq "ADDRESS_RADIX")
            {
25                 $mifAddressRadix = mifRadixFromText($2);
                dprint "mif -- address radix = $mifAddressRadix";
            }
            elsif($1 eq "DATA_RADIX")
            {
30                 $mifDataRadix = mifRadixFromText($2);
                dprint "mif -- data radix = $mifDataRadix";
            }
        }
        elsif($mifrecord =~ /^(.*):(.*)"$/ )
        {
35             dprint "mif -- $1 : $2";
            $mifAddress = mifValueByRadix($1,$mifAddressRadix) *
            $mifBytesPerData;
            $mifData = mifValueByRadix($2,$mifDataRadix);
40             dprint "mif -- $mifAddress $mifData";
            for($i = 0; $i < $mifBytesPerData; $i++)
            {
45                 dprint "mif -- ",$i + $mifAddress,":",$mifData&0xff;
                $hash{$mifAddress + $i} = $mifData & 0xff;
                $mifData >>= 8;
            }
        }
50     }
    dprint "mif hash keys";
    foreach $i (sort(keys(%hash)))
    {
55         dprint"$i:$hash{$i}";
    }

    return %hash;
}
60

```

5

10

```

# -----
# hash2dat(hashRef,width,lanes,lane,info)
#
15 # Returns giant string ready to be written to a file.
#
sub hash2dat
{
20   my $bytesRef = shift;
   my $width = shift;
   my $lanes = shift;
   my $lane = shift;
   my $info = shift;

25   my $addressLow;
   my $addressHigh;
   my @addresses;

   my $address;
30   my $addressRange;
   my $addressStep;
   my $bytesPerData;
   my $dataFormat;
   my $depth;
35   my $result = "";

   my $i;
   my $v;
   my $line;
40   my $bytesPerLine;
   my $bytesThisLine;

   @addresses = sort ( ( $a <=> $b ) keys(%$bytesRef) );
   $addressLow = $addresses[0];
45   $addressHigh = $addresses[$#addresses] + 1;
   $addressRange = $addressHigh - $addressLow;

   $bytesPerData = ceil($width / 8);

50   $addressStep = $lanes * $bytesPerData;
   $dataFormat = "%0" . $bytesPerData * 2 . "X";

   $depth = log(ceil($addressRange / $addressStep)) / log(2);
   $depth = ceil($depth);
55   $depth = 1 << $depth;

   #
   # Print the MIF file header
   #

60   addTo \$result, sprintf("@%08X",$addressLow / $addressStep);

   $line = "";

```

09880105-051201

```

$bytesThisLine = 0;
$bytesPerLine = 16;
for($address = $addressLow + $lane * $bytesPerData ; $address <
5 $addressHigh ; $address += $addressStep)
{
    $v = 0;
    for($i = 0; $i < $bytesPerData; $i++)
    {
10         $v += $$bytesRef[$address + $i] << ($i * 8);
    }
    $line .= sprintf($dataFormat,$v) . " ";
    $bytesThisLine += $bytesPerData;
    if($bytesThisLine >= $bytesPerLine)
15     {
        addTo \$result, $line;
        $line = "";
        $bytesThisLine = 0;
    }
20     addTo \$result, $line if $line ne "";

    # Is this needed? the old converter tacked on an extra 00...
    addTo \$result, "00" x $bytesPerData;

25     return $result;
}

# -----
30 # hash2mif(hashRef,width,lanes,comments,lane,info)
#
# Returns giant string ready to be written to a file.
#
sub hash2mif
35 {
    my $bytesRef = shift;
    my $width = shift;
    my $lanes = shift;
    my $comments = shift;
40     my $lane = shift;
    my $info = shift;

    my $addressLow;
    my $addressHigh;
45     my @addresses;

    my $address;
    my $addressRange;
    my $addressStep;
50     my $bytesPerData;
    my $dataFormat;
    my $depth;
    my $result = "";

55     my $mifAddress;
    my $i;
    my $v;

    @addresses = sort( ({a <=> $b} keys(%$bytesRef)));
60     $addressLow = $addresses[0];
    $addressHigh = $addresses[$#addresses] + 1;
    $addressRange = $addressHigh - $addressLow;

```

09330106-061201

```

$bytesPerData = ceil($width / 8);

$addressStep = $lanes * $bytesPerData;
$dataFormat = "%0" . $bytesPerData * 2 . "X";

5
$depth = log(ceil($addressRange / $addressStep)) / log(2);
$depth = ceil($depth);
$depth = 1 << $depth;

10
#
# Print the MIF file header
#

addTo \$result;
15
if($comments)          # MAX hates mif comments, so say --comments=0 if
you're MAX.
{
    addTo \$result, "/* This file generated by nios-convert */";
    addTo \$result, "/* $info */";
20
    addTo \$result, "/* " , dateTime() , " */";
    addTo
        \$result, "/* " ,          sprintf("0x%08x-
0x%08x", $addressLow, $addressHigh) , " */";
}
    addTo \$result;
25
    addTo \$result, "WIDTH=", $width, ";";
    addTo \$result, "DEPTH=", $depth, ";";
    addTo \$result;
    addTo \$result, "ADDRESS_RADIX=HEX;";
    addTo \$result, "DATA_RADIX=HEX;";
30
    addTo \$result;
    addTo \$result, "CONTENT BEGIN";
    addTo \$result;

    $mifAddress = 0;
35
    for($address = $addressLow + $lane * $bytesPerData ; $address <
$addressHigh ; $address += $addressStep)
    {
        $v = 0;
        for($i = 0; $i < $bytesPerData; $i++)
40
        {
            $v += $$bytesRef{$address + $i} << ($i * 8);
        }
        addTo \$result, sprintf(" %08X : $dataFormat;", $mifAddress++, $v);
    }
45

    addTo \$result;
    addTo \$result, "END;";
    addTo \$result;
    addTo \$result, "/* End of file */";

50
    return $result;
}

55
# -----
# parseArgs
#
# Given a list of arguments, return
# a hash where the keys and values
# are taken from those arguments of
60
# the form "--key=value". The hyphens
# disappear from the key name.

```

09880106-051201

```

#
# A command line switch of "--key"
# is equivalent to "--key=1".
#
5 # a special key named _argc contains
# a count of non-dash-dash arguments,
# and they are in the hash as {0}, {1},
# and so on.

10 sub parseArgs
    {
        my $arg;
        my $argVal;
        my $argc;
15     my %hash;

        $argc = 0;

20     while($arg = shift)
        {
            dprint "parseArgs: $arg";
            usage if $arg eq "--help";

25             if($arg =~ /^--/)
            {
                if($arg =~ /^--(.*)\=(.*)$/)
                {
                    $arg = $1;
                    $argVal = $2;
30                 }
                else
                {
                    $argVal = 1;
35                 }

                $hash{$arg} = $argVal;
            }
            else
            {
40                 $hash{$argc++} = $arg;
            }
        }

45     $hash{_argc} = $argc;

    return %hash;
}

50 # -----
# getSwitch(hashRef, switchName, defaultValue [, mustBeNumber])
#
# Look at a hash as returned by parseArgs, and
# give the value of the switch, or the defaultValue
55 # if it was not specified in the command line.

sub getSwitch
{
    my $hashRef = shift;
    my $switchName = shift;
    my $defaultValue = shift;
    my $mustBeNumber = shift;
60

```

09880106-061201

```

my $switchValue;

$switchValue = $$hashRef{$switchName};
$switchValue = $defaultValue if ($switchValue eq "");
5  $switchValue *= 1 if ($mustBeNumber);

return $switchValue;
}

10 # -----
# main

sub main
{
15  my %switches;
    my $lanes;
    my $width;
    my $sourceFileName;
    my $sourceFileNameBase;
20  my $sourceFormat;
    my $destFileName;
    my $destFileBase;
    my $destFormat;

25  my $sourceFile;          # complete contents
    my @destFile;           # complete contents, indexed by lane
    my $lane;

    %switches = parseArgs(@_);

30  $lanes = getSwitch(\%switches,"lanes",1);
    $width = getSwitch(\%switches,"width",16);
    $comments = getSwitch(\%switches,"comments",1);

35  #
    # Source name & format
    #

40  $sourceFileName = $switches{0};
    usage() if $sourceFileName eq "";

    if($sourceFileName =~ /^(.*)\.[^\.]*$/)
    {
45      $sourceFileNameBase = $1;
      $sourceFormat = $2;
    }

    #
50  # Dest name & format
    #

    $destFormat = "mif";
    if($switches{1})
55  {
        dprint "switch1 = $switches{1}";
        $destFileName = $switches{1};
        if($destFileName =~ /^(.*)\.[^\.]*$/)
        {
60          $destFileNameBase = $1;
          $destFormat = $2;
        }
    }
}

```

09330106-061201

```

$destFormat = getSwitch(\%switches,"oformat",$destFormat);
$destFileNameBase = ${sourceFileNameBase} if ($destFileName eq "");
$destFileName = "${destFileNameBase}.${destFormat}";

```

```

5  dprint "destFormat = ",$destFormat;
   dprint "destFileNameBase = ",$destFileNameBase;
   dprint "destFileName = ",$destFileName;
   dprint "sourceFileName = ",$sourceFileName;

```

```

10  $sourceFile = readFile($sourceFileName);
    die "Bad file $sourceFileName" if $sourceFile eq "";

```

```

    if($sourceFileName =~ /\.mif$/)
    {
15      %bytes = mif2hash($sourceFile);
    }
    else
    {
20      %bytes = srec2hash($sourceFile);
    }

```

```

    $sourceFile = ""; # done with source data, thankyou

```

```

25  #
    # Generate each lane of the result file,
    # switching by destFormat
    #

```

```

30  for($lane = 0; $lane < $lanes; $lane++)
    {
        if($destFormat eq "mif")
        {
            $destFile[$lane] =
35  hash2mif(\%bytes,$width,$lanes,$comments,$lane,
            "source file: $sourceFileName, lane $lane of
            $lanes");
        }
        elseif($destFormat eq "dat")
        {
40      $destFile[$lane] = hash2dat(\%bytes,$width,$lanes,$lane,
            "source file: $sourceFileName, lane $lane of
            $lanes");
        }
45      #
        # If we supported something besides mif, we'd add it here...
        # elsif...
    }

```

```

50  for($lane = 0; $lane < $lanes; $lane++)
    {
        if($lanes > 1)
        {
            $destFileName = $destFileNameBase . "_lane_" . $lane . "." .
55  $destFormat;
        }
        writeFile($destFileName,$destFile[$lane]);
    }

```

```

60

```

```

# -----
sub usage

```

09830106-061201


```
{
print <<EOP;
```

```
    nios-convert [options] sourceFile [destFile]
```

```
    sourceFile can be .srec or .mif
    destFile will get same name as sourceFile if omitted
```

```
    --lanes=x      : break up into multiple output files, with _lane_0 ..
    _lane_(x-1) appended
    --width=x      : set output width to 8, 16, or 32
    --oformat=f    : format can be mif or dat
    --comments=b   : comments in mif file enabled (1) or disabled (0).
    Default is enabled
```

```
"nios-convert"
```

nios-convert is a tool to convert files between several formats. The formats supported are S-record, mif, and dat. These are three formats used in Nios hardware and software development. S-records are used to download code to the Germs monitor. mif files are used to specify the contents of Nios ROM and RAM devices. dat files are used as data for Modelsim to simulate a Nios hardware design.

It is sometimes necessary to break up a file into individual "lanes"; if the --lanes option is used for more than 1 lane, then the result files will have the names "lane_0", "lane_1", &c, appended to them.

```
EOP
```

```
    exit(0);
}
```

```
main(@ARGV);
```

```
# end of file
```

09880106-061201

Nios-Convert.pl

#!/bin/sh
exec perl - "\$@" <<\ENDOFPERL
5 #!perl

We now use "nios-convert" in the HDK.
in the HDK, we can't rely on any standard Perl libraries.
... alas, not even "Strict":
10 #use Strict;

nios-convert

15 # -----
ceil(x)

standard math ceil

20 sub ceil
{
 \$x = shift;

 return int(\$x) if (\$x == int(\$x));
25 return int(\$x + 1);
}

addTo(stringRef,list)
30 sub addTo
{
 my \$stringRef = shift;

35 my \$i;

 for(\$i = 0; \$i <= \$#_; \$i++)
 {
 \$\$stringRef .= \$_[\$i];
40 }
 \$\$stringRef .= "\n";
}

45 # dprint(list...)

print only if gDebug

my \$gDebug = 0;
50 sub dprint
{
 my \$i;

55 return if (\$gDebug == 0);

 for(\$i = 0; \$i < 30; \$i++)
 {
 print shift;
60 }
 print "\n";
}

09330106-061201

```

# -----
# dateTime()
#
# returns a relatively nice date & time string
5 #
sub dateTime
{
    my ($sec,$min,$hour,$mday,$mon,$year,$yday,$isdet) =
10 localtime(time);
    $mon++;
    $year += 1900;

    my $d = sprintf("%04d.%02d.%02d",$year,$mon,$mday);
    my $t = sprintf("%02d:%02d:%02d",$hour,$min,$sec);
15
    return "$d $t";
}

20 # -----
# readFile(fileName)
#
# returns the complete file contents
#
25 sub readFile
{
    my $fileName = shift;
    my $bunch;
    my $result;
    my $did;
30
    if(open(FILE,$fileName))
    {
        binmode FILE; # Bite me, Windows! --dvb
        while(read(FILE,$bunch,32000))
        {
            $result .= $bunch;
        }
        close FILE;
40
    }

    return $result;
}

45 # -----
# writeFile(fileName,contents)
#
# creates new file and writes entire
50 # file contents. Return "ok" if so,
# or "" if not.
#
sub writeFile
{
55     my $fileName = shift;
    my $contents = shift;
    my $did;

    #
60     # Delete existing file, if any.
    #
    unlink ($fileName) if(-e $fileName);

```

```
$did = open(FILE, ">$fileName");
if($did)
```

```
{
    binmode FILE;                # Bite me, Windows! --dwb
    print FILE $contents;
    close FILE;
    return "ok";
}
```

```
return "";
}
```

```
# -----
# srec2hash(giantFileString)
#
# Given a giant string containing
# one entire srec file, return an
# associative array where each entry
# has a byte-address key, and its
# contents.
```

```
# This will of course be moderately
# gigantic, but who's counting?
#
# We just presume that RAM is cheap
# and plentiful and works, too.
#
```

```
sub srec2hash
```

```
{
    my $srecString = shift;
    my %hash;
```

```
    my $srecord;
    my $recordType;
    my $recordLength;
    my $recordChecksum;
    my $recordAddress;
    my $recordData;
    my $addressStringLength;
```

```
    foreach $srecord (split("\n", $srecString))
```

```
    {
        $srecord =~ s/\r//g;    # kill ^M's
        dprint "record $srecord";
        if($srecord =~ /^S([123])(...)(.*)...$/)    # an S record we
        can use
```

```
{
    $recordType = $1;
    $recordLength = hex($2)-1;
    $a = $3;
    $recordChecksum = $4;
```

```
    $recordLength -= $recordType + 2;
    $addressStringLength = ($recordType + 1) * 2;
    $recordAddress = hex(substr($a, 0, $addressStringLength));
    $recordData = substr($a, $addressStringLength);
```

```
    while(length($recordData))
    {
        $hash{$recordAddress} = hex(substr($recordData, 0, 2));
        $recordData = substr($recordData, 2);
    }
}
```

09830106-061201

```

                                $recordAddress++;
                                }
                        }
5   dprint "srec hash keys";
    foreach $i (sort(keys(%hash)))
    {
        dprint"$i:$hash($i)";
    }
10   return %hash;
    }

15   sub mifRadixFromText
    {
        my $mifRadix = shift;

20       return 10 if($mifRadix eq "UNS"
                    or $mifRadix eq "DEC");
        return 16;
    }

25   sub mifValueByRadix
    {
        my $mifData = shift;
        my $mifRadix = shift;

30       return hex($mifData) if($mifRadix == 16);
        return 1.0 * $mifData;
    }

35   # -----
    # mif2hash(giantFileString)
    #
    # Given a giant string containing
    # one entire mif file, return an
    # associative array where each entry
40   # has a byte-address key, and its
    # contents.
    #
    # This will of course be moderately
    # gigantic, but who's counting?
45   #
    # We just presume that RAM is cheap
    # and plentiful and works, too.
    #
    sub mif2hash
    {
50         my $srecString = shift;
        my %hash;

        my $mifrecord;
55         my $mifWidth;
        my $mifAddressRadix;
        my $mifDataRadix;
        my $mifBytesPerData;
        my $mifData;
60         my $mifAddress;
        my $i;

        $mifWidth = 8;

```

09880106-061201

```

        foreach $mifrecord (split("\n",$srecString))
        {
            $mifrecord =~ s/\r//g; # kill ^M's
5      dprint "mif record $mifrecord";

        # Recognize 4 kinds of lines:
        #   WIDTH=x;
        #   ADDRESS_RADIX=HEX/DEC/UNS;
10     #   DATA_RADIX=HEX/DEC/UNS;
        #   addr:data;
        # Ignore anything else. We don't even care about the "DEPTH".
        #
            $mifrecord =~ s/[ \t ]//g; # kill white space
15     if($mifrecord =~ /^(.*)\=(.*)"$/){
        {
            dprint "mif -- $1 = $2";
            if($1 eq "WIDTH")
            {
20                 $mifWidth = $2;
                $mifBytesPerData = int($mifWidth / 8);
            dprint "mif -- mifBytesPerData = $mifBytesPerData";
            }
            elsif($1 eq "ADDRESS_RADIX")
            {
25                 $mifAddressRadix = mifRadixFromText($2);
            dprint "mif -- address radix = $mifAddressRadix";
            }
            elsif($1 eq "DATA_RADIX")
            {
30                 $mifDataRadix = mifRadixFromText($2);
            dprint "mif -- data radix = $mifDataRadix";
            }
        }
        elsif($mifrecord =~ /^(.*):(.*)"$/){
35         {
            dprint "mif -- $1 : $2";
            $mifAddress = mifValueByRadix($1,$mifAddressRadix) *
            $mifBytesPerData;
            $mifData = mifValueByRadix($2,$mifDataRadix);
40         dprint "mif -- $mifAddress $mifData";
            for($i = 0; $i < $mifBytesPerData; $i++)
            {
45                 dprint "mif -- ",$i + $mifAddress,":",$mifData&0xff;
                $hash{$mifAddress + $i} = $mifData & 0xff;
                $mifData >>= 8;
            }
        }
    }
50     dprint "mif hash keys";
    foreach $i (sort(keys(%hash)))
    {
        dprint"$i:$hash($i)";
55     }

    return %hash;
}

60

```

5

10

15

20

25

30

35

40

45

50

55

60

```

# -----
# hash2dat(hashRef,width,lanes,lane,info)
#
# Returns giant string ready to be written to a file.
#
sub hash2dat
{
    my $bytesRef = shift;
    my $width = shift;
    my $lanes = shift;
    my $lane = shift;
    my $info = shift;

    my $addressLow;
    my $addressHigh;
    my @addresses;

    my $address;
    my $addressRange;
    my $addressStep;
    my $bytesPerData;
    my $dataFormat;
    my $depth;
    my $result = "";

    my $i;
    my $v;
    my $line;
    my $bytesPerLine;
    my $bytesThisLine;

    @addresses = sort ( ( { $a <=> $b } keys(%$bytesRef) );
    $addressLow = $addresses[0];
    $addressHigh = $addresses[$#addresses] + 1;
    $addressRange = $addressHigh - $addressLow;

    $bytesPerData = ceil($width / 8);

    $addressStep = $lanes * $bytesPerData;
    $dataFormat = "%0" . $bytesPerData * 2 . "X";

    $depth = log(ceil($addressRange / $addressStep)) / log(2);
    $depth = ceil($depth);
    $depth = 1 << $depth;

    #
    # Print the MIF file header
    #

    addTo \$result, sprintf("\%08X",$addressLow / $addressStep);

    $line = "";

```

```

    $bytesThisLine = 0;
    $bytesPerLine = 16;
    for($address = $addressLow + $lane * $bytesPerData ; $address <
5      $addressHigh ; $address += $addressStep)
    {
        $v = 0;
        for($i = 0; $i < $bytesPerData; $i++)
        {
            $v += $$bytesRef[$address + $i] << ($i * 8);
10        }
        $line .= sprintf($dataFormat,$v) . " ";
        $bytesThisLine += $bytesPerData;
        if($bytesThisLine >= $bytesPerLine)
        {
15            addTo \$result, $line;
            $line = "";
            $bytesThisLine = 0;
        }
20    }
    addTo \$result, $line if $line ne "";

    # Is this needed? the old converter tacked on an extra 00...
    addTo \$result, "00" x $bytesPerData;

25    return $result;
}

# -----
30 # hash2mif(hashRef,width,lanes,comments,lane,info)
#
# Returns giant string ready to be written to a file.
#
sub hash2mif
35 {
    my $bytesRef = shift;
    my $width = shift;
    my $lanes = shift;
    my $comments = shift;
40    my $lane = shift;
    my $info = shift;

    my $addressLow;
    my $addressHigh;
45    my @addresses;

    my $address;
    my $addressRange;
    my $addressStep;
50    my $bytesPerData;
    my $dataFormat;
    my $depth;
    my $result = "";

55    my $mifAddress;
    my $i;
    my $v;

    @addresses = sort( {$a <=> $b} keys(%$bytesRef));
60    $addressLow = $addresses[0];
    $addressHigh = $addresses[$#addresses] + 1;
    $addressRange = $addressHigh - $addressLow;

```



```

$bytesPerData = ceil($width / 8);

$addressStep = $lanes * $bytesPerData;
$dataFormat = "%0" . $bytesPerData * 2 . "X";

5
$depth = log(ceil($addressRange / $addressStep)) / log(2);
$depth = ceil($depth);
$depth = 1 << $depth;

10
#
# Print the MIF file header
#

addTo \$result;
15
if($comments)          # MAX hates mif comments, so say --comments=0 if
you're MAX.
{
    addTo \$result, "/* This file generated by nios-convert */";
    addTo \$result, "/* $info */";
20
    addTo \$result, "/* " . dateTime() . " */";
    addTo
        \$result, "/* " . sprintf("0x%08x-
0x%08x", $addressLow, $addressHigh) . " */";
}
    addTo \$result;
25
    addTo \$result, "WIDTH=", $width, ";";
    addTo \$result, "DEPTH=", $depth, ";";
    addTo \$result;
    addTo \$result, "ADDRESS_RADIX=HEX;";
    addTo \$result, "DATA_RADIX=HEX;";
30
    addTo \$result;
    addTo \$result, "CONTENT BEGIN";
    addTo \$result;

    $mifAddress = 0;
35
    for($address = $addressLow + $lane * $bytesPerData ; $address <
$addressHigh ; $address += $addressStep)
    {
        $v = 0;
        for($i = 0; $i < $bytesPerData; $i++)
40
        {
            $v += $$bytesRef{$address + $i} << ($i * 8);
        }
        addTo \$result, sprintf(" %08X : $dataFormat;", $mifAddress++, $v);
    }
45

    addTo \$result;
    addTo \$result, "END;";
    addTo \$result;
    addTo \$result, "/* End of file */";

50
    return $result;
}

55
# -----
# parseArgs
#
# Given a list of arguments, return
# a hash where the keys and values
# are taken from those arguments of
# the form "--key=value". The hyphens
# disappear from the key name.
60

```

```

#
# A command line switch of "--key"
# is equivalent to "--key=1".
#
5  # a special key named _argc contains
# a count of non-dash-dash arguments,
# and they are in the hash as {0}, {1},
# and so on.

10 sub parseArgs
    {
        my $arg;
        my $argVal;
        my $argc;
15     my %hash;

        $argc = 0;

20     while($arg = shift)
        {
            dprint "parseArgs: $arg";
            usage if $arg eq "--help";

25             if($arg =~ /^--/)
            {
                if($arg =~ /^--(.*)\=(.*)$/)
                {
                    $arg = $1;
                    $argVal = $2;
30                 }
                else
                {
                    $argVal = 1;
35                 }

                $hash{$arg} = $argVal;
            }
            else
40             {
                $hash{$argc++} = $arg;
            }
        }

45     $hash{_argc} = $argc;

        return %hash;
    }

50 # -----
# getSwitch(hashRef, switchName, defaultValue [, mustBeNumber])
#
# Look at a hash as returned by parseArgs, and
# give the value of the switch, or the defaultValue
55 # if it was not specified in the command line.

sub getSwitch
    {
        my $hashRef = shift;
60     my $switchName = shift;
        my $defaultValue = shift;
        my $mustBeNumber = shift;
    }

```

```

my $switchValue;

$switchValue = $$hashRef{$switchName};
$switchValue = $defaultValue if ($switchValue eq "");
5   $switchValue *= 1 if ($mustBeNumber);

return $switchValue;
}

10  # -----
    # main

sub main
{
15    my %switches;
    my $lanes;
    my $width;
    my $sourceFileName;
    my $sourceFileNameBase;
20    my $sourceFormat;
    my $destFileName;
    my $destFileBase;
    my $destFormat;

25    my $sourceFile;          # complete contents
    my @destFile;             # complete contents, indexed by lane
    my $lane;

    %switches = parseArgs(@_);

30    $lanes = getSwitch(\%switches, "lanes", 1);
    $width = getSwitch(\%switches, "width", 16);
    $comments = getSwitch(\%switches, "comments", 1);

35    #
    # Source name & format
    #

40    $sourceFileName = $switches{0};
    usage() if $sourceFileName eq "";

    if($sourceFileName =~ /^(.*)\[^\.\]*$/)
    {
45        $sourceFileNameBase = $1;
        $sourceFormat = $2;
    }

    #
50    # Dest name & format
    #

    $destFormat = "mif";
    if($switches{1})
55    {
        dprint "switch1 = $switches{1}";
        $destFileName = $switches{1};
        if($destFileName =~ /^(.*)\[^\.\]*$/)
        {
60            $destFileNameBase = $1;
            $destFormat = $2;
        }
    }
}

```

09820106 "061201
T02T90" 9010885

```
$destFormat = getSwitch(\%switches,"oformat",$destFormat);
$destFileNameBase = ${sourceFileNameBase} if ($destFileName eq "");
$destFileName = "${destFileNameBase}.${destFormat}";
```

```
5  dprint "destFormat = ", $destFormat;
    dprint "destFileNameBase = ", $destFileNameBase;
    dprint "destFileName = ", $destFileName;
    dprint "sourceFileName = ", $sourceFileName;
```

```
10      $sourceFile = readFile($sourceFileName);
      die "Bad file $sourceFileName" if $sourceFile eq "";
```

```
if($sourceFileName =~ /\.mif$/)
```

```
{
%bytes = mif2hash($sourceFile);
}
```

```
20         %bytes = srec2hash($sourceFile);
        }
```

```
$sourceFile = ""; # done with source data, thankyou
```

```
25      #
      # Generate each lane of the result file,
      # switching by destFormat
      #
```

```
30     for($lane = 0; $lane < $lanes; $lane++)
        {
            if($destFormat eq "mif")
```

```
35 hash2mif(\%bytes,$width,$lanes,$comments,$lane,  
           "source file: $sourceFileName, lane $lane of  
           $lanes");
```

```

    elif($destFormat eq "dat")
    {
40         $destFile[$lane] = hash2dat(\%bytes,$width,$lanes,$lane,
            "source file: $sourceFileName, lane $lane of
$lanes");
    }
}

```

```
45      #
      # If we supported something besides mif, we'd add it here...
      # elsif...
      }
```

```

50         for($lane = 0; $lane < $lanes; $lane++)
            {
                if($lanes > 1)
                {
                    $destFileName = $destFileNameBase . "_lane_" . $lane . ". " .
55 $destFormat;
                }
                writeFile($destFileName,$destFile[$lane]);
            }

```

60

```
# -----
sub usage
```

```
{  
print <<EOP;
```

```
    nios-convert [options] sourceFile [destFile]
```

5

```
    sourceFile can be .srec or .mif  
    destFile will get same name as sourceFile if omitted
```

```
10    --lanes=x      : break up into multiple output files, with _lane_0 ..  
    _lane_(x-1) appended  
    --width=x      : set output width to 8, 16, or 32  
    --oformat=f    : format can be mif or dat  
    --comments=b   : comments in mif file enabled (1) or disabled (0).  
    Default is enabled
```

15

```
"nios-convert"
```

20

```
nios-convert is a tool to convert files between several  
formats. The formats supported are S-record, mif, and dat.  
These are three formats used in Nios hardware and software  
development. S-records are used to download code to the  
Germs monitor. mif files are used to specify the contents  
of Nios ROM and RAM devices. dat files are used as data for  
Modelsim to simulate a Nios hardware design.
```

25

```
It is sometimes necessary to break up a file into individual  
"lanes"; if the --lanes option is used for more than 1 lane,  
then the result files will have the names "lane_0", "lane_1",  
&c, appended to them.
```

30

```
EOP
```

35

```
    exit(0);  
    }
```

```
    main(@ARGV);
```

40

```
# end of file
```

09880106 061201
T021201

Ptf_Update.pl

#!/contrib/bin/perl

5 # Just a wrapper to call the "main function" in this here module,
so that you can do it from the command-line:
use wiz_utils;

&PTF_Translate_Old_Version (@ARGV);

10

09880106 061201

```

srec2mif.pl

#!/contrib/bin/perl

5  # -----
  # srec2mif
  # usage: srec2mif source-file
  # dvb \ Altera \ 2000

10

    my $file = shift;          # name of file to read
    my $outFile;
    my $a;
15    my $recordType;
    my $recordLength;
    my $recordAddress;
    my $recordData;
    my $recordChecksum;
20    my $residue;
    my $i;

    $outFile = $file . ".mif";
    if($file =~ /^(.*)\.srec$/)
25    {
        $outFile = $1 . ".mif";
    }

    open(FILE,"<$file") or die "could not write to $file";
30    open(OUTFILE,">$outFile") or die "could not write to $outFile";

    # changed output file to make it MAX+Plus2 Friendly,
    # specifically c style comments are not handled and
    # UNS should be DEC.
35    print OUTFILE <<EOP;

    WIDTH=16;
    DEPTH=512;

40    ADDRESS_RADIX=DEC;
    DATA_RADIX=HEX;

    CONTENT BEGIN
    EOP
45    while($a = <FILE>)
        {
            if($a =~ /^S([123])(..)(.)(..)$/)      # an S record we
can use
                {
50                    $recordType = $1;
                    $recordLength = hex($2)-1;
                    $a = $3;
                    $recordChecksum = $4;

55                    $recordLength -= $recordType + 2;
                    my $addressStringLength = ($recordType + 1) * 2;
                    $recordAddress
=
                    hex(substr($a,0,$addressStringLength));
                    $recordData = substr($a,$addressStringLength);

60                    if($residue)
                        {
                            $recordAddress--;

```

```

        $recordData = $residue . $recordData;
    }

    $recordLength = length($recordData) & ~3;
5    $i = 0;
    while($i < $recordLength)
    {
        printf OUTFILE " %5d : %s%s;\n", $recordAddress/2,
10         substr($recordData, $i+2, 2),
        substr($recordData, $i, 2);
        $recordAddress += 2;
        $i += 4;
    }
    $residue = substr($recordData, $i);
15    }

    print OUTFILE <<EOP;

20    END;
    EOP

    # And end of this file, too!
25

```

09880106-061201
T02T90-90T08860


```

srec2sim.pl

#!/contrib/bin/perl

5  #
  $HELP_STRING = <<EOT;
  # srec2sim.pl          (C) dvb 2000 Altera
  #
  #
10 # Translates any Nios S-record into Verilog memory-initialization
  # (.dat) files.
  #
  # --- OVERVIEW ---
  #
15 # Let's suppose you have a program --perhaps even a C-program-- and you
  # want to simulate a Nios system running that program.
  #
  # As you know, the Nios PBM-wizard generates, among other things, a
  # test bench for the Nios System Module you specify.  How handy.  But
20 # how do you get it to run a given program?  Wouldn't it be great
  # if your simulated Nios System Module were connected to a simulated
  # memory device with -your program- already loaded into it?
  #
  # Yep, that'd be cool.
25 #
  # This utility converts any compiled Nios program (any S-record)
  # into four .dat-files.  These dat-files are suitable directly for use
  # with the provided behavioral memory model "Demo_Ext_Ram_Sim_Model.v"
  #
30 # That behavior model contains four individual byte-wide memory arrays
  # (one array for each byte-lane of a 32-bit-wide memory bank).  At
  # simulation-start, each byte-lane is initialized with contents from a
  # file of a particular, fixed name.  The four fixed file names are:
  #
35 #     external_ram_lane_0.dat    -- Initializes bits  0..7  of memory.
  #     external_ram_lane_1.dat    -- Initializes bits  8..15 of memory.
  #     external_ram_lane_2.dat    -- Initializes bits 16..23 of memory.
  #     external_ram_lane_3.dat    -- Initializes bits 24..31 of memory.
  #
40 # This utility emits these four files as output.  It takes, as input, an
  # S-record which encodes the program or data you want these files
  # to contain.
  #
  # -- USAGE --
45 #
  # perl srec2sim.pl  [-b <mem-base-address>]  <srec-file-name>
  #
  # <mem-base-address> :  Target memory base-address in the simulated system.
  #                       For the Nios demo kit, the external (main) memory
50 #                       is mapped at address 0x40000.  That would be the
  #                       number you'd enter here.  The default value is
  #                       zero (0).
  #
  #
55 EOT

60 # -----
  # Globals accessible to everyone...
  #
  my @memory;          # byte by byte memory snarf.

```

09280106 "051201"

```
my $memoryLow = 999999999;
my $memoryHigh = 0;

5  my $file      = "";
    my $mem_base = 0;

    while ($arg = shift(@ARGV))
    {
10     $mem_base = eval (shift(@ARGV)), next if $arg =~ /^-b$/i;

        die ($HELP_STRING) if $file;
        $file = $arg;
    }
15  die ($HELP_STRING) if !$file;

    # This comment gets stuck-in at the top of all the generated output
20  # files:
    $OUTPUT_HEADER= <<EOT;
    // External memory model initialization file.
    // contents translated from the S-record file:
    //
25  //     $file
    //
    // These contents were relocated for a memory with
    // a chip-select base-address at $mem_base.
    //
30  // This file was created by the utility script "srec2sim.pl." You
    // can use this script to convert any S-record (compiled program)
    // into a group of four memory initialization files.
    //
    // These files are designed for use with the module
35  //
    //     Demo_Ext_Ram_Sim_Model
    //
    // That module is just a simple behavioral (NOT timing-accurate) model
    // of the 32 bits x 64K asynchronous SRAM which comes on the Nios
40  // demo kit board.
    //
    // That module gets its initial data from four files. This is one of those
    // files.
    //
45  // For more information about simulating a Nios core running a compiled
    // program, see the comments atop "srec2sim.pl"
    //
    EOT

50  #-----
    # EmitDATFiles
    #
    # This just uses all the globals inherited
    # from main. Why a subroutine? So we can
55  # add a different one, perhaps, for MIF
    # or other spewers.

    sub EmitDATFiles
    {
60     my $lanes = 4;
        my $outFileBase;
        my $outFile;
        my $address;
```

09880106 061201
T02T90" 90T08860

```
my $lane;
my $lineCount;

$outFileBase = $file;
5 $outFileBase = $1 if ($file =~ /^(.*)\.srec$/);
$outFileBase .= ".dat";

#
# Emit a file for each lane
10 #

for($lane = 0; $lane < $lanes; $lane++)
{
    $outFile = "external_ram_lane_$lane" . ".dat";
15 $outFile = $outFileBase . $lane;

    open (OUTFILE, ">$outFile") or die "could not write to $outFile";
    printf OUTFILE $OUTPUT_HEADER;

20 printf OUTFILE "@%04x\n", int(($memoryLow - $mem_base) / $lanes);

    $lineCount = 0;

    for($address = $memoryLow + $lane; $address < $memoryHigh; $address
25 += $lanes)
    {
        printf OUTFILE "%02x ", $memory[$address];
        $lineCount++;
        if($lineCount >= 16)
        {
30             print OUTFILE "\n";
            $lineCount = 0;
        }
    }
    printf OUTFILE "\n00\n";          # why? the example had it.
    close (OUTFILE);
    }
40 }

45

my $a;
my $recordType;
my $recordLength;
50 my $recordAddress;
my $recordData;
my $recordChecksum;
my $residue;
my $i;
55

#
# Step 0. Open input file
#

60 open(FILE, "<$file") or die "could not read from to $file";

#
# Step 1. Read every S-Record we can understand
```

```

#         into @memory, or $memory[], if you prefer.
#
while($a = <FILE>)
5      {
      if($a =~ /^S([123])(..)(.*)($/))          # an S record we can use
      {
          $recordType = $1;
          $recordLength = hex($2)-1;
10         $a = $3;
          $recordChecksum = $4;

          $recordLength -= $recordType + 2;
          my $addressStringLength = ($recordType + 1) * 2;
15         $recordAddress = hex(substr($a,0,$addressStringLength));
          $recordData = substr($a,$addressStringLength);

          # recordAddress is the where the first byte
          # of recordData goes. recordData is ascii hex pairs,
20         # byte by byte

          while($recordData =~ /^(..)(.*)$/)
          {
              $memory[$recordAddress] = hex($1);
25         $memoryLow  = $recordAddress if $recordAddress <
$memoryLow;
              $memoryHigh  = $recordAddress if $recordAddress >
$memoryHigh;

              $recordData = $2;
              $recordAddress++;
30         }
          }
35         }

        printf ("Read S-records from %X to %X\n", $memoryLow, $memoryHigh);
        printf (" Relocating to memory at base address %X.\n", $mem_base)
            if $mem_base;

40         #
        # Step 2. Spew out the memory file in some format.
        #

        EmitDATFiles();
45

# And end of this file, too!

```

09880106-061201
102150-90108860

vhdl_simulation.pl

#Copyright (C) 2000-2001 Altera Corporation

5 #####
 #main program begins here.

my \$help_string = <<END_OF_HELP;

10 vhdl_simulation.pl

This program converts nios 1.1 verilog simulation files to vhdl simulation files. It has been provided as a service to customers who design with vhdl exclusively. It converts verilog simulation files generated by the nios 1.1 kit into equivalent vhdl files. This program is NOT a general verilog to vhdl converter.

20 It is the belief of the Altera Nios group that the hdl world is split roughly evenly between vhdl and verilog users. Designers who wish to incorporate intellectual property (IP) from external sources into their designs will quickly find themselves in need of bilingual HDL tools. Such tools exist and we enthusiastically endorse them. We have had a good experience with a bilingual simulator license from modelsim technology. You can contact them at <http://www.model.com/>.

25 To run this program, you first need to generate nios 1.1 verilog suitable for simulation. To do this, you must exit the Nios System Builder MegaWizard and hand edit your <system.ptf> file. Under the system WIZARD_SCRIPT_ARGUMENTS you should add the line.

30 do_build_sim = "1";

35 It is important that you put this line in the WIZARD_SCRIPT_ARGUMENTS of the system and not WIZARD_SCRIPT_ARGUMENTS for a particular module. Then open up the MegaWizard and re-generate your system. From now on, whenever you re-generate your system, an updated simulation directory will be created.

40 You will also need a perl distribution which you can obtain on the web from active perl. If you don't already have perl 5.0 or greater running on your computer, you can get it from:

<http://www.activestate.com/Products/ActivePerl/Download.html>

45 Then from a command line cd to your nios project/sim directory and type:

perl vhdl_simulation.pl <nios_system_test_bench.v>

e.g. to build a vhdl sim model of the 32 bit reference design:

50 perl d:/temp/bin/vhdl_simulation.pl ref_32_system_test_bench.v

55 The program will then take all the verilog files referred to by ref_32_system_test_bench.v and convert them into equivalent vhdl files.

END_OF_HELP

60 if (-e \$ARGV[0])
 {
 &Convert_Simulation_Files(\$ARGV[0]);
 }

```

else
{
    print $help_string;
}
5
#####
#
# V2VHD and accompanying functions takes a verilog file and converts it to a
vhd file.
10
#
#####
#
# This works with most synthesizable verilog code.
# Here is what is not supported and known ways to work around it:
15
# 0) Verilog numbers must not be more than 32 bits wide
# 1) No Procedure/Tasks allowed.
# 2) Asynchronous set/resets must follow clk in always declaration
#     i.e. always @(posedge reset or posedge clk) not supported,
#         always @(posedge clk or posedge reset) is.
20
# 3) Asynchronous set/resets must be first declared i.e.
#     always @(posedge clk or posedge reset)
#     begin
#         if (reset)
#             //asynchronous stuff here.
25
#         else
#             //synchronous statements here
#         endif
# 4) Params may not define width. i.e. output [Width - 1: 0] foo not supported
# 5) Params must be able to convert to types of natural or string
30
# 6) Parameters must be set using defparam Instance.param=value not #value
Instance
# 7) Instantiation must connected by Instance (.port (connection)) not
Instance (connection)
# 8) No case statements allowed. Use a lot of if statements instead.
35
# 9) verilog and my code is case sensitive, vhdl isn't.
#10) don't name a wire/module/instance a vhdl reserved word.
#11) Integers are forced to be 32 bits wide
#12) verilog /funky_@!#naming_convention_with_slash not supported
#13) timing statements are not supported i.e. a <= #23 b; will be treated as a
40
<= b;
#14) $readmemb, $readmemh and $write are the only special $variables supported
#15) It's currently possible to name a wire = tmp_logic later on in the module
or some
#     other name that get_exclusive name doesn't know about yet. The solution
45
is to
#     run through first and gather up all known names then do Exclusive name on
the known name
#     set
#####
50
#Ways to make code more readable
# 1) Put comments back in
# 2) Find a way to convert boolean to std_logic_vector.
# 3) Make wire assignments a list with keys = rhs, value = lhs
#     then when we make a new wire we can look for key rhs before making a new
55
wire.
# 4) Find a better way to concatenate /reduce names, sign extend names
# 5) Fix condition where wire tmp_logical is declared later on after wire name
is
#     created.
60
# Known Bugs:
# Is_real should replace equivalences on parenthesis names.
#####
#

```

```

# read_file ($file_index, $complete_filename)
#
# read_file returns the contents of the file named $complete_filename.
# Sounds like a cakewalk except that Verilog has a means of including files
5 # much like c++ does.
#
# So everytime read_file sees:
#
# `include foo
10 #
# it calls:
# &read_file ($file_index+1,foo);
#
# and sticks the result directly in its string, which it returns when it
15 # reaches the end of its file. (Recursive functions are like that).
#
# This file uses a global list called $include_list.
# $include list is used to ensure that infinite `include loops don't happen
# see check_for_infinite_include_loops
20 #####
#
sub read_file
{
    my ($file_index,$complete_filename,$path) = @_;
25     my $file_contents;

    $complete_filename =~ tr|\\|\/|;
    $path =~ tr|\\|\/|;

    #warn "path,filename $path,$complete_filename\n";
    $path = "." unless $path;
    open ($file_index,"<$path\/$complete_filename") ||
        die "Unable to open $complete_filename, $!";
    while(<$file_index>)
35     {
        if (0)#(s/^s*\`include\s+\\"(.*)\\"//)
        {
            my $fn = $1;
            #print "in file $complete_filename, found include, '$1\n";
40             $fn =~ tr|\/|\\|;
            if ($fn !~ /^(\\\\\\\\)|(\w\:))/?
            {
                #full path name isnt specified. Use previous directory location.
                $fn = $path.$fn;
45             }
            $include_list{$complete_filename} .= "," if
($include_list{$complete_filename});
            $include_list{$complete_filename} .= $fn;
            #print "include_list for $complete_filename is
50             ".$include_list{$complete_filename}.
            #" check value is $complete_filename\n";

            &check_for_infinite_include_loops($fn,$complete_filename);
            $file_contents .= &read_file($file_index+1,$fn,$path);
55         }
        else
        {
            $file_contents .= $_;
60         }
    }
    close($file_index);
    return ("\n$file_contents\n");
}

```

```

#####
#
# check_for_infinite_include_loops
#
5 # I'm getting sick of recursive functions. Here's another one.
#
# This one hunts down the tree structure in $included_file(list) and makes sure
# that files don't refer back on themselves via `includes.
#
10 # It is okay for multiple `includes of the same file as long as that file
# doesn't include something which includes something which includes the original
# file.
#
# To check, we put a stake in the ground ($check_value) and traverse the
15 `include files
# If we ever see our stake again, we know we've walked in a circle.
#
#####
#
20 sub check_for_infinite_include_loops
{
    my ($included_file,$check_value) = @_;
    #print "c_f_i_i_l, if, $included_file, cf, $check_value\n";
    if (!$include_list{$included_file})
25 {
        return;                                     #never heard of this file
    }
    before                                         #no infinite loop here.
}

30 foreach $key (split(/\./,$include_list{$included_file}))
    # I've seen this guy somewhere before
    {
        # make sure I'm not
        circling.
        if ($key eq $check_value)
35 {
            die "ERROR: FOUND INFINITE INCLUDE LOOP!\nFILE $check_value
EVENTUALLY INCLUDES ITSELF!\n";
        }
        else
40 {
            #print "checking $key against $check_value\n";
            #no smoking gun yet, check what this file includes.
            #keep the same $check_value;
            &check_for_infinite_include_loops($key,$check_value);
45        }
    }
    return;
}
sub Kill_Comments
50 {
    my ($line) = @_;

    #Kill multi_line_comments
    my $tmp_line;
55 while ($line =~ s|^(\.*?)\\/\*(.*?)\\*\\/||s)
    {
        $tmp_line .= $1;
    }
    $tmp_line .= $line;
    #Kill single_line_comments
60 $line = "";
    while ($tmp_line =~ s|^(\.*?)\\/\.*?\n|\n|s)
    {

```



```

    $line .= $1;
}
$line .= $tmp_line;
return ($line);
5  }
sub Process_Comments
{
    my ($line) = @_;
    # Comments
10    # VHDL does not have a means for making multi line comments
    # That means that we have to convert the lines to verilog single line
    comments (//)

    while ($line =~ s|^(*?)\\/(.*?)\\*\\/|s|s)
15    {
        my $commented_line = $2;
        #There better not be nested comments in here!
        die "BAD COMMENT, $commented_line, NESTED COMMENTS!"
            if ($commented_line =~ /\/*\\/);
20
        # Convert all single line comments (//) in the comment to (/-/ ) so we
        don't
        # confuse ourselves later.
        while ($commented_line =~ s|\\/(\\/|\\/\\-\\/|)|{;})
25
        $commented_line = join ("\\n\\/",split(/\\n/, $commented_line));

        #####
        # Right now we just crush multi-line comments. We could get fancier
30    later
        # If you wanted to keep these comments, uncomment the next line.
        # $line .= "\\n/$commented_line\\n"
    }

35    #####
    # Now we just have single comments left. For these, we can just convert
    # the verilog single line comment // to the vhd1 single line comment --.
    # We also need to convert all ticked statements, e.g. (`define, `ifdef,
    `endif, etc.)
40    # to their non-ticked counterparts, (define, ifdef, endif, etc.) And since
    we
    # later split commands on semi-colon(;) Crush ; so that we won't have to
    worry about it.

45    # crush single-line comments
    while ($line =~ s|^(*?)\\/(.*?)\\n(.*?)|s|s)
    {
        my $comment = $2;
        my $rest = $3;
50        #warn "found single line comment named $comment, $1\\n";
        #Keep comments with special word "exemplar" in them for Leonardo
        Spectrum.
        #But convert comment character to vhd1 comment character so that we don't
        #get stuck in an infinite loop.
55
        if ($comment =~ /^\\s*exemplar/i)
        {
            #warn "putting $comment back into line\\n";
            $line .= "\\-\\-$comment\\n";
60        }
        $line .= $rest;

        #####

```

```

    # Just kill comments for now
    # while ($comment =~ s/(\`|\\;)//){;}
    # $line .= "\-\\-$comment";
}
5   return($line);
}

#####
10  #
    # Process_Verilog_Directives takes verilog code as its sole string argument.
    # It then processes all things in the file that have ` associated with it except
    # for `include
    # such ` things include
15  # `define foo 3
    # `ifdef, `else, `endif
    # `foo
    # It returns an equivalent verilog string devoid of all `marks. It replaces
    # all `foo with its definition if defined. (No error message is printed if foo
20  # has not been defined). It does the correct thing on `ifdef, `else, `endif
    # statements, (including nested ifdefs).
    #####
    #
    sub Process_Verilog_Directives
25  {
        my ($line) = @_;
        my $defined_line = "";

        undef %defined;
        undef %definition_of;
        my %defined;
        my %definition_of;
        my @nested_ifdefs = (1); #start off including code.
30
        my $I_Should_Keep_This_Part;

        $line = " ".$line; #a space is added so that first time through, split
        /\/, we pass through
        #all of the gates and emerged unscathed as the first part of $defined_line;
40
        #print "line is $line\n";
        foreach $tick (split (/\\`/s,$line))
        {
            $I_Should_Keep_This_Part = eval (join ('&&', @nested_ifdefs));
            #print ("tick is $tick, isktip is $I_Should_Keep_This_Part, nifdef = "
45  #.join ('|',@nested_ifdefs). "\nundefined line now is $defined_line -----
\n");
            die "unmatched `endif" if ((scalar (@nested_ifdefs)) == 0);
            if ($tick =~ /^ifdef\s+(\S+)(.*)/s)
50  {
                $defined{$$1} = 0 unless ($defined{$$1});
                $I_Should_Keep_This_Part = ($defined{$$1} &&
$I_Should_Keep_This_Part);
                push (@nested_ifdefs,$I_Should_Keep_This_Part);
                $defined_line .= $2 if ($I_Should_Keep_This_Part);
55  next;
            }

            if ($tick =~ /^else\s+(.*)/s)
60  {
                die ("saw `else before `ifdef") unless ((scalar (@nested_ifdefs) >
1));
                $I_Should_Keep_This_Part = pop (@nested_ifdefs);

```

09880106-061201 102190-9018860

09880106-061201
102150-90108860

```

    $I_Should_Keep_This_Part = (! $I_Should_Keep_This_Part) && eval (join
('&&', @nested_ifdefs));
    push (@nested_ifdefs, $I_Should_Keep_This_Part);
    $defined_line .= $1 if ($I_Should_Keep_This_Part);
5      next;
  }

  if ($tick =~ /^endif\s+(.*)/s)
  {
10      die ("saw `endif` before `ifdef` in $line\n") unless ((scalar
(@nested_ifdefs) > 1));
    pop (@nested_ifdefs);
    $I_Should_Keep_This_Part = eval (join ('&&', @nested_ifdefs));
    #print "endif says isktip is $I_Should_Keep_This_Part,nif ".join
15    ('|', @nested_ifdefs);
    $defined_line .= $1 if ($I_Should_Keep_This_Part);
    next;
  }

20  if ($tick =~ /\Adefine\s+(\S+)\s*(\S*)\s*$/m)
  {
    if ($I_Should_Keep_This_Part)
    {
      #print "definition of $1 is $2, vpp pass is $vpp_pass";
      #die "$1 is defined in 2 places" if ($defined{$1});
      $defined{$1} = 1;
      $definition_of{$1} = $2;
      $tick =~ s/^(.*?)\n(.*)/$2/s;
      $defined_line .= $tick;
25      }
    }
    next;
  }

  if ($tick =~ /^(\S+)(.*)/s)
  {
35      $tick = $definition_of{$1}.$2;
      $defined_line .= $tick if ($I_Should_Keep_This_Part);
      next;
    }
  }

40  if ($tick =~ /^s+/s)
  {
    #this should only be the first one.
    $defined_line .= $tick if ($I_Should_Keep_This_Part);
45      next;
  }
}

die "missing `endif`" if ((scalar (@nested_ifdefs)) > 1);

50  return ($defined_line);
}

sub date_time
{
55  my ($sec, $min, $hour, $mday, $mon, $year, $yday, $isdet) = localtime(time);
    $mon++;
    $year += 1900;

    my $d = sprintf("%04d.%02d.%02d", $year, $mon, $mday);
60    my $t = sprintf("%02d:%02d:%02d", $hour, $min, $sec);

    return "$d $t";
}
```

```
#####
#
# Verilog_Number_To_Bit_String
5 #
# Verilog_Number_To_Bit_String takes a verilog number of the form 5'hA
# and turns it into a quoted bit string "01010".
#####
#
10 sub Verilog_Number_To_Bit_String
{
    my ($verilog_number) = @_;
    my $integer_value = 0;
    my $width;

15     $verilog_number =~ s/^\s*(.*?)\s*$/$1/;

    die "ERROR NO WIDTH SPECIFIED FOR VERILOG NUMBER $verilog_number\n"
        unless ($verilog_number =~ s/^(\\d+)\\'//);

20     $width = $1;
    die "width is greater than 32 bits. I intend to fix this, but for now, you
are out of luck\n"
        if ($width > 32);
    # If there is an 'x' or 'z' in the number, return
    # a bitstream of x or z with width $width

    if ($verilog_number =~ /[zx]/i)
    {
30         my $tmp_string = $1 x $width;
        return ("\"$tmp_string\"");
    }

    if ($verilog_number =~ /^b([0-1]+)$/i)
    {
35         foreach $bit (split (//,$1))
        {
            $integer_value = $integer_value * 2;
            $integer_value += 1 if ($bit == 1);
        }
    }

    if ($verilog_number =~ /^d([0-9]+)$/i)
    {
45         $integer_value = $1;
    }

    if ($verilog_number =~ /^o([0-7]+)$/i)
    {
50         $integer_value = eval("0".$1);
    }

    if ($verilog_number =~ /^h([0-9a-f]+)$/i)
    {
55         $integer_value = eval("0x".$1);
    }

    die ("ERROR Verilog_Number_To_Bit_Size_And_Bit_String:\n"
        . "NUMBER $verilog_number NOT UNDERSTOOD!")
        if ($integer_value eq "");

60     if ($integer_value >= 2**32)
    {

```

```

#warn " verilog_number $verilog_number int_value ($integer_value) is
bigger than 2**32\n";
my @bit_array;
while ($integer_value > 0)
5   {
        unshift (@bit_array, ($integer_value % 2));
        $integer_value = $integer_value >> 1;
    }
my $bit_string = join ("", @bit_array);
10  #warn "test way to do it yields $bit_string\n";
}

#Orion, warn if number is >= than 2>>$width

15  my $return_string;
my $mask_value;
foreach $mask_bit (reverse(0..($width-1)))
{
    if ($integer_value & (1 << $mask_bit))
20    {
        $return_string .= "1" ;
    }
    else
    {
25        $return_string .= "0" ;
    }
}
#return ("\'$return_string\'") if ($width == 1);
return ("\"$return_string\"");
30 }

sub VN2BS {return ( &Verilog_Number_To_Bit_String(@_));}

sub Process_Concatenation
35 {
    my ($string,
        @Module_Info) = @_;
    my (
40        $Width_List,
        $Signal_List,
        $pWire_Assignments,
        $Equivalence_List) = @Module_Info;

    my ($begin, $middle, $end) = &Count_Parentheses($string, '\{', '\}');
45    #warn "PC: middle ($middle)\n";
    #warn %$Width_List;
    #warn "PC: end\n";
    die "ERROR Process_Concatenation, NO VALUE BETWEEN SQUIGGLY BRACKETS in
($string)\n"
50    if ($middle eq "");
    while ($middle =~ s/\{|\}\//g){;} #e.g. {a,b,c,{a,e},f,g} -> {a,b,c,a,e,f,g}

    my ($expanded_array, $width) =
    &Expand_Array_Of_Bit_Vectors_Into_Separated_Bits("", $middle, @Module_Info);
55    #warn "P_C, return_width is $return_width, width is $width\n";
    $string = "$begin";
    if (scalar(split (/\/, $expanded_array)) == 1)
    {
60        my $tmp_string = $expanded_array;
        while ($tmp_string =~ s/\'\/\'//){;}

        #Turn indexed bits into bit array of size 1
        while ($tmp_string =~ s/[\s*(\d+)\s*]/\[1:$1\]/s){;}
    }
}

```

00000106 001201

```

    $string .= $tmp_string;
}
else
{
5   $string .= " std_logic_vector\' ";

    $string .= "(";
    $string .= $expanded_array;
    $string .= " \)$end";
10  }
    #warn "PC: Concatenation = $string\n";
    return (&Replace ("Concatenation = $string",$width,@Module_Info));
}

15  #####
    #
    # Expand_Array_Of_Bit_Vectors_Into_Separated_Bits
    #
    # Does exactly what it says. e.g.
20  # &Expand_Array_Of_Bit_Vectors_Into_Separated_Bits (A(3 DOWNT0 2),"0010")
    # returns "A[3],A[2],\'0\',\'0\',\'1\',\'0\'"
    # We convert brackets to VHDL Parentheses at the end of the module
    #####
25  sub Expand_Array_Of_Bit_Vectors_Into_Separated_Bits
    {
        my ($separator,
            $Comma_Separated_String,
30         @Module_Info) = @_;

        my ($Width_List,
            $Signal_List,
            $pWire_Assignments,
35         $Equivalence_List) = @Module_Info;

        my $string = "";
        my $width = 0;

40         foreach $name (split(/\s*\,\s*/s,$Comma_Separated_String))
        {
            $name =~ s/^\s*(.*?)\s*$/$1/s;
            #warn "eabvisb name ($name)\n";

45             $name = &V2VHD_Equation($name, @Module_Info);

            #Convert Bit String into bits, i.e. "10110" => \'1\',\'0\',\'1\',\'1\',\'0\'
            if (&Replace_Equivalences($name,$Equivalence_List) ==~
50         /^(\"|\')([01XZ]+)\1$/i)
            {
                foreach $bit (split(//,$2))
                {
                    $string .= "$separator " if ($string);
55                 $string .= "\'$bit\'";
                    $width++;
                    #warn " eabvisb bit_string bit is $bit, width is $width\n";
                }
            }
            else
            {
                my ($left,$right);
60

```

```

($name,$left,$right) = &Vector_Range($name,$Width_List);

if ($left eq "") {$left = &Width_Of($name,$Width_List);}
if ($right eq "")
5   {
    $name = &Add_Intermediate_Signal("tmp_$name" =
$name",$left,@Module_Info);
    $left = eval ($left - 1);
    $right = 0;
10   }

    $left = eval($left);$right = eval($right);
    foreach $index (&Order($left,$right))
    {
15     $width++;
     $string .= "$separator\n\t" if ($string);
     $string .= "$name\[ $index\]";
     #warn " eaobvisb ($name($index)$separator) added to string, width
is $width \n";
20   }
    }

    return($string,$width);
25 }

#####
#
# Vector_Order
30 # Inputs, $left, $right
#
# Vector_Order (0,4) returns (0 TO 4)
# Vector_Order (3,1) returns (3 DOWNTO 0)
#####
35 #

sub Vector_Order
{
40   my ($left_index,$right_index) = @_;
   return "($left_index)"
   if ($right_index eq "");
   if (($left_index >= $right_index) || ($right_index == 0))
   {
45     return "($left_index DOWNTO $right_index)";
   }
   else
   {
     return "($left_index TO $right_index)";
50   }
}

#####
#
# Order
55 # Inputs, $left, $right
#
# Order (0,4) returns array (0,1,2,3,4)
# Order (3,1) returns array (3,2,1)
#####
60 #

sub Order
{
   my ($left,$right) = @_;

```

00000106.061201

0930106-061201

```
my @Vector_Array;
die "ERROR Order: LEFT VALUE ($left) NOT A NUMBER\n"
    unless ($left =~ s/^\s*(\d+)\s*$/\1/s);
die "ERROR Order: RIGHT VALUE ($right) NOT A NUMBER\n"
    unless ($right =~ s/^\s*(\d+)\s*$/\1/s);

    if ($right > $left){@Vector_Array = ($left..$right);}
    else{ @Vector_Array = reverse ($right..$left);}
    return (@Vector_Array);
}

#####
# Width_Of
#
# Returns the width of a variable, verilog number, or vhdl bit_stream.
# eg. Width_Of(4'h2) = 4; Width_Of "01010" = 5;
# Returns "" if the width is not known.
#
#####
sub Width_Of
{
    my ($var, $Width_List, $pParameter) = @_;

    #KILL SPACES! KILL! KILL!
    $var =~ s/^\s*(.*?)\s*$/\1/s;

    #A verilog number is nice enough to tell you its width
    #at the very beginning. Usually a pain, but in this case it is great
    if ($var =~ /^(\d+)\'[bdoh]([\d+a-fxz])/i)
    {
        #warn " WIDTH_OF found verilog number $var, returned width of $1\n";
        return ($1);
    }

    #Count the bits in a vhdl bit stream
    if ($var =~ /^(\\"|\')([01XZ]+)\1/i)
    {
        my $width = scalar (split (//,$2));
        #warn " WIDTH_OF found vhdl number $var, returned width of $width\n";
        return ($width);
    }

    my ($name,$left,$right) = &Vector_Range($var, $Width_List, $pParameter);
    #warn "Width_Of $var after Vector_Range, left,right -> $left, $right\n";
    return "" if ($left eq ""); #Not Known;
    return "$left" if ($right eq ""); #Known, but not a vector.
    return (abs($left - $right) + 1); #vector width arithmetic
}

#####
# Vector_Range
#
# Returns the name and (left and right value) of the vector range for a
# verilog or vhdl vector.
# if
# reg [4:0] foo;
# &Vector_Range (foo [3:2]) = (foo,3,2)
# &Vector_Range (foo) = (foo,4,0)
# &Vector_Range (foo[2]) = (foo,2,2)
#####
sub Vector_Range
{
    my ($var, $Width_List, $pParameter) = @_;
```


00000106-0612001
T02T90-90T08860

```
my $name;
my $left;
my $right;

5   $var =~ s/^\s*(.*?)\s*$/s;

    #If our var is foo [3:2] return (foo,3,2)
    #We're not sophisticated enough to deal with (>1)-dimensional
    #vectors. If you need a multi-dimensional vector width,
10   #there is a keyboard in front of you, start typing.

    if ($var =~ /\s*\[(.*?)\]/s)
    {
        #warn " Width_Of brackets, inside brackets is $2\n";
15     $name = $1;
        ($left,$right) = split /\s*:\s*/s,$2;
        $left = eval($left);
        $right = $left if ($right eq "");
        $right = eval($right);
20     return ($name,$left,$right);
    }

    #If our var is foo (3 DOWNT0 2) return (foo,3,2)
    if ($var =~ /\s*\((.*?)\)/s)
    {
25     #warn " Width_Of brackets, inside brackets is $2\n";
        $name = $1;
        my $index = $2;
        ($left,$right) = split /\s*(DOWN)?TO\s*/s,$index;
30     $left = eval($left);
        $right = $left if ($right eq "");
        $right = eval($right);
        return ($name,$left,$right);
    }

35     #No indecies are specified, that means use the whole
    #variable. Fortunately, we have already saved all
    #variable widths in %$Width_List
    ($left,$right) = split /\s*,\s*/s,$Width_List{$var};
40     $name = $var;
    return ($name, $left,$right);
}

#####
45 #
# Replace_Parameters
# Replaces all text in $val with $$pParameter{text}
# Currently not being used

50 sub Replace_Parameters
{
    my ($val, $pParameter) = @_;
    while ($val =~ s/^(.*?)(\b[a-zA-Z_\]\w*)(.*)$/s)
    {
55     last if ($2 =~ /integer/i);
        my $param_substitution = $$pParameter{$2};
        die "ERROR PARAMETER SUBSTITUTION: PARAMETER $2 NOT DEFINED\n"
            if ( $param_substitution eq "");
        $val .= $param_substitution.$3;
60     }
    return (eval ($val));
}
}
```

```

#####
#
# Get_Port_Name_Direction_And_Type
# Takes a vhdl port string and returns Name, Direction, Type and Vector Range
5 # i.e. SIGNAL data_from_cpu : OUT STD_LOGIC_VECTOR
# returns ("data_from_cpu", "OUT", "STD_LOGIC_VECTOR(31 DOWNT0 0)")
#
sub Get_Port_Name_Direction_And_Type
10 {
    my ($string) = @_;
    my $possible_variables = 'SIGNAL|VARIABLE|SHARED\s+VARIABLE';
    if ($string =~ /^\\s*($possible_variables)\\s+(\\w+)\\s*:\\s*(\\w+)\\s*(.*)$/is)
    {
        my ($name,$direction,$type) = ($2,$3,$4);
15         return ($name,$direction,$type);
    }
    else
    {
        die "ERROR Get_Port_Name_Direction_And_Type, IMPROPER STRING $string\\n";
20     }
}

#####
#
# Declare_Signal
#
# takes, $name,$pSignal_Width as arguments, returns
# SIGNAL $name : STD_LOGIC_VECTOR ($left DOWNT0 $right)
# or
30 # SIGNAL $name : STD_LOGIC_VECTOR ($left TO $right)
# depending on pSignal_Width

sub Declare_Signal
35 {
    my ($name,$pSignal_Width) = @_;
    my ($left,$right) = split (/\\./,$pSignal_Width{$name});
    die "Declare_Signal, bad inputs ($name,$left,$right)\\n"
        if (($name =~ /\\W/) || ($left =~ /\\D/) || ($right =~ /\\D/));
    my $Signal_Declaration;
40     if ($left < $right)
    {
        $Signal_Declaration = "SIGNAL $name : STD_LOGIC_VECTOR ($left TO
$right);";
    }
    else
45     {
        $Signal_Declaration = "SIGNAL $name : STD_LOGIC_VECTOR ($left DOWNT0
$right);";
    }
50     #warn "DECLARE_SIGNAL RETURNINGG $Signal_Declaration\\n";
    return ($Signal_Declaration);
}

#####
#
# Convert_Signals_To_Shared_Variable
#
# Takes a string and a pointer to Signal_List.
# Converts every word(\\b([a-zA-Z]\\w*)\\b) in the Signal_List from a SIGNAL
# to a SHARED VARIABLE. Returns an array of all words changed.
60 # This function is only needed to handle blocking statements.

sub Convert_Signals_To_Shared_Variable
{

```

```

my ($line,$Signal_List) = @_;
die "ERROR Convert_Signals_To_Shared_Variable, Signal_List IS NULL\n"
    if (!$Signal_List);
my @return_array;
5   while ($line =~ s/\b([a-zA-Z]\w*)\b//)
    {
        my $variable = $1;
        push (@return_array,$variable);
        my $tmp_Signal_List = $$Signal_List{$variable};
10      die "ERROR VARIABLE CONVERSION: of ($variable) FAILED
($tmp_Signal_List)\n"
        unless ($tmp_Signal_List =~ s/^(\\s*)(SIGNAL|SHARED VARIABLE)/$1SHARED
VARIABLE/s);
        #warn " CSTSV, signal was $$Signal_List{$variable} now is
15      $tmp_Signal_List\n";
        $$Signal_List{$variable} = $tmp_Signal_List;
    }
}

20 sub Process_Register_Assignment
{
    my ($line,@Module_Info) = @_;
    my ($Width_List,
        $Signal_List,
25      $pWire_Assignments) = @Module_Info;
    my $lhs;
    my $rhs;
    my $lhs_width;
    my $after_line;

30      #Process $readmem(b|h), $write, etc
    #warn "line is $line\n";
    if ($line =~ s/^(\\s*)(\\$.*)\\s*/$2/s)
    {
35      return ($1.&Process_Dollar_Verilog_Statements($line,@Module_Info));
    }

    #Search for delay patterns
    $line =~ s/(.*)\\#\\s*\\S*(.*)/$1 $2/s;
    # "#" delay kind of broken due to differences between languages
    # representation of delays, so just forget about them for now
    if (0) #($line =~ s/(.*)\\#\\s*(.*)/$1/s)
    {
45      my $wait_amount = $2;
        my ($tmp,$delay_amount,$rest_of_line);

        if ($wait_amount =~ /^\\(\\/)
        {
50      ($tmp,$wait_amount,$rest_of_line) = &Count_Parentheses($wait_amount);
            $line .= $rest_of_line;
        }
        else
        {
55      $wait_amount =~ s/^(\\S+)(.*)/$1/s;
            $line .= $beginning_of_line.$2;
        }
        #If there is only a delay statement, e.g. (#32;) return "wait for 32 ns;"
        if ($line =~ /^\\s*\\;/s)
        {
60      return ("WAIT FOR $wait_amount ns;");
        }
        else
        {

```

```

    $after_line = "AFTER $wait_amount ns";
}

}

5
#There are two kinds of verilog assignments that can be made
# 1) blocking "=" assignment made immediately
# 2) non-blocking "<=" assignment made after all other items in that
#    timestamp are equated

10
if ($line =~ /^(.*) (\<|=|)=) (.*)$/s)
{
    my ($lhs,$operator,$rhs) = ($1,$2,$3);
    my $tmp_pWire_Assignments;
15
    my $line = &V2VHD_Equation_Wrapper (" $lhs =
$rhs", $Width_List, $Signal_List, \ $tmp_pWire_Assignments);
    #warn "P_R_A: After Wrapper wire assignments are $ $pWire_Assignments\n";

    if ($operator eq "\=")
20
    {
        $line =~ s/^(.*) \<=(.*)$/ $1: $2/s;
        #warn "found blocking statement ($lhs$operator$rhs) lhs is ($lhs)\n";

        #####
        # BLOCKING STATEMENTS are tricky.
        # Verilog has this concept of "blocking and non-blocking" statements.
        # VHDL has module scoped SIGNALS that can only be non-blocking and
        # process scoped VARIABLES that must be blocking. To convert
        # verilog blocking statements into an equivalent VHDL blocking
25
        statement,
        # we must first convert the lhs of the blocking assignments within
        our process
        # into VARIABLES. At the end of our process, we transform all our
        VARIABLE
        # names into temporary variable names and assign the original SIGNAL
35
        names
        # equal to the temporary variable names. Perhaps an example will
        keep my
        # confusing explanation from spinning your head any longer.
40
        #
        # always @(posedge clk)
        #   a = a + 1;
        #
        # will become:
45
        # PROCESS BEGIN
        # WAIT UNTIL clk = "1";
        #
        # number_1_1_bits_wide := "1";
        --all machine generated equations
        #   a_2_bits_wide := std_logic_vector' ('0', variable_a(0)) +
50
        --must precede assignment and be
        #
        # std_logic_vector' ('0',
        number_1_1_bits_wide(0)); --of type VARIABLE
        #   variable_a := a_2_bits_wide(0 DOWNT0 0);
        #   a <= variable_a;
55
        # END PROCESS
        #
        #####

        #####
60
        # First, we convert any machine generated wire assignments to
        variable
        # assignments within the line since they need to be evaluated before
        # the "effective immediate"(ly) variable assignment.

```

09880106 061201

```
# in the above example, number_1_1_bits_wide and a_2_bits_wide are
# "machine generated".
```

```
my $machine_generated_commands = "";
#warn "tmp_WA ($tmp_pWire_Assignments)\n";
my (@blocking_commands) = split (/\\/, $tmp_pWire_Assignments);
foreach $block_command (@blocking_commands)
{
    $block_command =~ s/^(.*?)\\<(\\=.*?)$/$1:$2;/s;
    &Convert_Signals_To_Shared_Variable($1,$Signal_List);
    $machine_generated_commands .= "$block_command";
}
$line = $machine_generated_commands.$line;
```

```
#####
# Now we'd like to translate all left hand side arguments in the
process
# into tmp names. Unfortunately, we only know what's in this line,
not
# what is in the entire process. So for each chip design, we put a
non-vhdl
# sentence that says "Please Convert $signal_name To A Variable" in
$line.
# We'll do the conversion later. Pretty hacky, I know.
```

```
$lhs =~ s/[.*?\\]/_/s; #do not convert memory indexes;
while ($lhs =~ s/\\b([a-zA-Z]\\w*)\\b//)
{
    $line .= "Please Convert $1 To A Variable";
}
```

```
}
else {$$pWire_Assignments .= $tmp_pWire_Assignments;}
```

```
if ($after_line ne "")
{
    $line =~ s/\\;\\s*$//s;
    return ("$line $after_line;");
}
```

```
else
{
    return ("$line");
}
```

```
}
else
{
    return "";
}
```

```
}
sub Get_Exclusive_VHDL_Name
{
    my ($name,$List) = @_;
```

```
$name =~ tr/A-Z/a-z/;
#First make $name into a vhdl acceptable name
#warn "eatme 2, $name\n"
#if ($GLOBAL_DEBUG);
#Say goodbye to everything that is not a word character in name.
while ($name =~ s/\\W+//){}
```

```
#Convert spaces to underscores
while ($name =~ s/\\s+/_/s){}
```

09880106.061201

```

#But not too many underscores, vhdl will not let you have two underscores
in your name or
#any underscores at either end
5   while ($name =~ s/(\_){2,}/\_/s){;}

    while ($name =~ s/^\_+(.*?)\_+$/1/){;}

    #vhdl does not like names to begin with numbers
10   $name =~ s/^\(d)/number_1/;
    #warn "eatme\n";
    #warn "IN GETXNAME\n"
    #if ($GLOBAL_DEBUG);
    while ($$List{$name} ne "")
15   {
        #warn "G_E_N name $name already taken. Trying tmp_$name\n"
        #if ($GLOBAL_DEBUG);
        $name = "tmp_$name";
    }
20   #warn "G_E_N name $name not taken.\n"
    #if ($GLOBAL_DEBUG);
    #Keep Place holder at this name.
    $$List{$name} = "Taken";
    return ($name);
25 }
#####
#
# Convert_Integers_To_Std_Logic_Vector
#
30 # Convert_Integers_To_Std_Logic_Vector takes left, operator, right as inputs.
# If left/right are integers, it converts them to a bit vector.
# If the widths of left or right is not known, it sets the unknown width equal
to the known width.
# (Perhaps it should not do this?)
35 #
# It does not use the operator now, but it maybe useful later.

sub Convert_Integers_To_Std_Logic_Vector
40 {
    my ($left,$operator,$right,$width_List,$Equivalence_List) = @_;
    my $left_width = &Width_Of($left,$width_List);
    my $right_width = &Width_Of($right,$width_List);

    #Check that widths are equal if both are known.
45   if (($left_width) && ($right_width))
    {
        if ($left_width != $right_width)
        {
            my $tmp_left = &Replace_Equivalences ($left,$Equivalence_List);
            my $tmp_right = &Replace_Equivalences ($right,$Equivalence_List);
50           #warn ("WARNING EXPRESSION ($left $operator $right)\n"
                #."WIDTH OF $tmp_left ($left_width) != WIDTH OF $tmp_right
                ($right_width)!\n");
        }
55     }
    else #Warn if neither width is known.
    {
        if (!(($left_width || $right_width))
60         {
            die ("CV2SLV VECTOR WIDTHS ($left_width) ($right_width) UNKNOWN FOR
                EXPRESSION ARITHMETIC OPERATION ($2) ($3) ($4)\n");
        }
    }
}

```

```

else #If only one width is known, convert the unknown width.
{
    if ($left_width) #right width is unknown.
    {
        5      if ($right =~ /\s*(\d+)\s*/s) #number => verilog decimal number
with width $left_width
        {
            #warn ("right width is unknown, left width is $left_width\n".
            #"sending $left_width\`d$1 to VN2BS");
            10      $right = &VN2BS("$left_width\`d$1");
        }
    }
    else #left width is unknown.
    {
        15      if ($left =~ /\s*(\d+)\s*/s) #number => verilog decimal number
with width $right_width
        {
            #warn ("left width is unknown, right width is $right_width\n".
            #"sending $right_width\`d$1 to VN2BS");
            20      $left = &VN2BS("$right_width\`d$1");
        }
    }
}
}
25 #Now determine the proper width.
#take max
my $max_width = $right_width;
$max_width = $left_width
    if ($left_width > $right_width);
30
return ($max_width,$left,$right);
}

#####
35 #
# Replace_Equivalences
# Replaces all tmp names in Equivalence List. Since some names
# In Equivalence_List refer to other names in the same Equivalence_List,
# Replace_Equivalences is Recursive. See big comment before sub V2VHD_Equation
40 # for an explanation of the bigger picture.

sub Replace_Equivalences
{
    45      my ($string,$Equivalence_List) = @_;
    my $new_string;
    my $replacement_value;
    while ($string ne "")
    {
        50      if ($string =~ s/^\([W\"'\d]+\)/s){$new_string .= $1;next;}
        if ($string =~ s/^\(w+\)/s)
        {
            my $word = $1;
            $replacement_value = $$Equivalence_List{$word};
            if ($replacement_value eq "")
            55      {
                $new_string .= $word;
            }
            else
            {
                60      my $new_replacement_value
&Replace_Equivalences($replacement_value,
                        $Equivalence_List);

```

```
#####
#parentheses get replaced if there is additional replacement levels
#inside the parentheses. e.g. verilog statement
# (a+b) => (binary) => parentheses.
# When replacing,
# parentheses => (binary) => (a+b)
#
# (a) => parentheses
# When replacing
# parentheses => a //no parentheses
```

```

    if ( ($word =~ /parentheses/i) &&
($$Equivalence_List{$replacement_value}))
    {
        $new_replacement_value = "\($new_replacement_value\)";
    }
    $new_string .= $new_replacement_value;
}
}
return ($new_string);
}
```

```
sub Vestigual_Replace_Equivalences
```

```
{
    my ($string,$tmp_Equivalence_List) = @_ ;
    my %Equivalence_List = %$tmp_Equivalence_List;

    my $values_were_changed = 1;
    while ($values_were_changed)
    {
        $values_were_changed = 0;
        foreach $key (keys(%$Equivalence_List))
        {
            my $value = $$Equivalence_List{$key};

            #Put parentheses around value if key was parenthesized
            #And it needs it
            if ( ($key =~ /Parentheses/) && ($$Equivalence_List{$value}))
            {
                $value = "\($value\)";
            }

            while ($string =~ s/\b$key\b/$value/)
            {
                #warn "key is $key\n value is $value\n string is $string\n";
                $values_were_changed = 1;
                undef $$Equivalence_List{$key};
            }
        }
    }

    #Now crush parentheses around single objects
    #while ($string =~ s/\(((\s[a-zA-Z"\'\[\]\s\w"\']*)))/$1/s){;}

    #warn "Replace_Equivalences, done, string is $string\n";
    return ($string);
}
```

```
#####
#
# ReadMem takes a verilog $readmem(b|h)(<filename>,<memory>)
# instruction and then hardwires _memory_ to the values in _filename_.
# This isn't as flexible as a true conversion of readmem would be (<filename>
# could be parameterizable and be different for multiple instances of the
# same module. In the future, this should be replaced with a real vhd1
```

09880106-061201

equivalent, but it is so much easier to parse filenames with perl than with
vhdl

```

5  sub ReadMem
    {
        my ($mem_radix,
            $dat_filename,
            $mem_name,
            @Module_Info) = @_;

10     my ($Width_List,
        $Signal_List,
        $pWire_Assignments,
        $Equivalence_List) = @Module_Info;

15     my $return_string;
        my ($left,$right,$up,$down) = split (/\\s*\\,\\s*/s,$$Width_List{$mem_name});
        my ($width) = &Width_Of($mem_name,$Width_List);
        open (DAT,"< $dat_filename") or
20         open (DAT,"< nios_system_sim\\/$dat_filename") or

            warn "\\nWarning, cannot open $dat_filename ($!)\n";

        my $dat_contents;
        while (<DAT>)
        {
            if (s/^(.*?)\\//.*/$1/){;}
            $dat_contents .= $_
                unless (/^\\s*/s);
30        }
        close (DAT);

        my @Address_Data_Pairs = split(/\\@/, $dat_contents);
        foreach $address_data (@Address_Data_Pairs)
        {
35            my ($address,@data) = split (/\\s+/s,$address_data);
            #warn "address is $address. data is @data\n";
            my $integer_address = eval ("0x$address");
            foreach $datum (@data)
            {
40                if ($mem_radix eq "b")
                {
                    $datum = "\\\"$datum\\\"";
45                }
                else
                {
                    $datum = &VN2BS("$width\\'$mem_radix$datum");
50                }

                $return_string .= "$mem_name($integer_address) <= $datum;\n";
                $integer_address++;
            }
        }
55     return ($return_string);
    }

#####
#
60 # Process_Dollar_Verilog_Statements
#
# Converts verilog $readmemb, $readmemh and $write statements
# to the appropriate vhdl equivalent

```

09860106-061201
T00T90-90T0990

```

#
sub Process_Dollar_Verilog_Statements
{
    my ($equation,@Module_Info) = @_ ;
5
    my ($Width_List,
        $Signal_List,
        $pWire_Assignments,
        ) = @Module_Info ;
10
    my $return_string ;

    if ($equation =~
15 / ^ \s* \$readmem (b|h) \s* ( \s* " \s* ( \S+ ) \s* " \s* \, \s* ( \w+ ) \s* ) \s* \; \s* $ / is )
    {
        my ($mem_radix,$dat_filename,$mem_name) = ($1,$2,$3) ;
        return (&ReadMem($mem_radix,
20             $dat_filename,
             $mem_name,
             @Module_Info
             )
            );
    }
    if ($equation =~ / ^ \$write \s* ( \s* " (.*?) " \s* \, \s* (.*?) \s* ) \s* \; / is )
25 {
        my ($quote,$values) = ($1,$2) ;

        my @values_array = split (/ \s* \, \s* / s, $values) ;
        my @string_array = ("") ; #put null in first spot and index from 1 ..
30 $string_length
        my $string_length = 0 ;
        while ($quote)
        {
            $string_length++ ;
            if ($quote =~ s/ ^ ( [ \% \\ ] \w ) // )
            {
35                 die "ERROR \\\$equation ONLY \%c IS CURRENLTLY SUPPORTED AS DATA
TYPE!\n"
                    unless ($1 =~ / \%c / i) ;
                push
40 (@string_array,"character'val(CONV_INTEGER(".shift(@values_array)."))") ;
            }
            else
            {
                $quote =~ s/ ^ ( . ) // ;
                push (@string_array,"\'$1\'") ;
45            }
        }
        # make a string signal of width $string_length
50 my $string_name = &Get_Exclusive_VHDL_Name("string_name",$Width_List) ;
        $$Signal_List{$string_name} = "SIGNAL $string_name : STRING(1 TO
$string_length) ;" ;
        $$Width_List{$string_name} = "1,$string_length" ;

55        #warn "string_array is @string_array\n" ;
        foreach $index (1..$string_length)
        {
            $return_string .= "
60 $string_array[$index] ;\n" ;
        }
        $return_string .= "    write(output,$string_name) ;\n" ;

        return($return_string) ;

```

09880106 061201 102190 90108860

```

    }

    die "ERROR Process_Dollar_Verilog_Statements, STATEMENT ($equation) NOT
UNDERSTOOD\n";
5   }

sub V2VHD_Equation_Wrapper
{
10   my ($equation,@Module_Info) = @_;

    my ($Width_List,
        $Signal_List,
        $pWire_Assignments,
        $Equivalence_List) = @Module_Info;

15   my %E_List;

    my $pE_List = \%E_List;
    $pE_List = $Equivalence_List
20     if ($Equivalence_List);

    my $string = &V2VHD_Equation(@_, $pE_List);
    $string = &Replace_Equivalences($string, $pE_List);

25   #Undef the tmp names so that we can use them again next time
    #Do not undef if $Equivalence_List was non-null because
    #we've been recursively called.
    if (!$Equivalence_List)
    {
30       foreach $key (keys(%pE_List))
        {
            #undef ($$Width_List{$key});
            $$Width_List{$key} = "";
        }
35   }
    #kill std_logic_vector' on lhs concatenations
    while ($string =~ s/^\s*std_logic_vector\'//s){;}

    #convert assignment operator to <=
40   $string =~ s/\={1}\s*/\<=\s/;

    #warn " Final Answer: $string\n";
    return ($string);
}
45 #####
#
# Replace
#
# Replace replaces a complicated expression with a simple exclusive name.
50 # See V2VHD_Equation comment.
# e.g.
# &Replace ("foo = a + b", 4,@Module_Info);
# would get an exclusive name for foo, set
# $Equivalence_List{&Get_Exclusive_Name(foo,$Width_List)} = a + b; set
55 # $Width_List{<the exclusive name for foo>}= 4
# and return the exclusive name for foo so that V2VHD_Equation can substitute
it
# into equations.

60 sub Replace
{
    my ($equation,
        $width,

```

```

    @Module_Info) = @_;

    my ($Width_List,
        $Signal_List,
5      $pWire_Assignments,
        $Equivalence_List) = @Module_Info;

    #die "ERROR Replace: width is ($width) in equation ($equation)\n"
    #unless $width;
10    my ($replacement_name,@tmp_replacement_value) = split
        (/\\s*\\={1}\\s*/s,$equation);
    my $replacement_value = join (" = ",@tmp_replacement_value);
    $replacement_name =
15    &Get_Exclusive_VHDL_Name("$replacement_name",$Width_List);
    $$Equivalence_List{$replacement_name} = $replacement_value;
    $$Width_List{$replacement_name} = $width;
    return ($replacement_name);
}

20 #####
#
# Find_In_Order
#
# searches $equation for each term in a "|" separated regexp.
25 # searches in order and returns the first exp that matches.
# returns a \ escaped string so that $equation can be
# patterned matched.

sub Find_In_Order
30 {
    my ($equation,$regexp) = @_;
    $regexp =~ s/([^\|])$/$1|/s;

    my $tmp_regexp = $regexp;
35 my $exp;
    while ($tmp_regexp =~ s/^(.*?[^\\])\\|//)
    {
        $exp = $1;
        #warn "FIO looking for ($exp)\n";
40         if ($equation =~ /($exp)/)
        {
            $exp = $1;
            #warn "FIO found ($exp) IN EQUATION ($equation)\n";
            last;
45         }
    }
    die "ERROR Find_In_Order: ($regexp) NOT FOUND IN ($equation)\n"
        unless ($exp ne "");
    $exp =~ s/(\\W)/\\$1/g;
50    return ($exp);
}

#####
55 #
# V2VHD_Equation
#
# Verilog and VHDL have similar operators, but of course they are
# different because as Buddah says, "Life is Suffering".
#
60 # Fortunately, since the operators do pretty much the same thing, we
# dont have to evaluate too many verilog expressions. For most
# operators, we just need to translate verilog arithmetic into vhd1
# arithmetic.

```

```

#
# Verilog and vhd1 handle numbers differently, so we need to convert
# numbers from verilog (32,4'hF,0) into their corresponding vhd1
# std_logic_vector bit strings "10000","1111","00". We also need to
5 # determine the bit width of the number. We do that based upon the
# closest known width operating on the number. i.e. (foo[4:0] == 0)
# means 0 probably has width of 5.
#
# VHDL really like "types" and that gets us in a little trouble. A
10 # naive translation of the verilog expression:
#         wire counter_ne_zero = !(counter[3:0] == 0)
# Would be:
#         SIGNAL counter_ne_zero: STD_LOGIC;
#         counter_ne_zero <= NOT (counter(3 DOWNT0 0) = "0000")
15 #
# Unfortunately, the result of (counter(3 DOWNT0 0) = 0) is type
# boolean, not std_logic. If there were an easy way to type change a
# boolean to std_logic, it would be easy to say:
#         counter_ne_zero <= NOT (To_std_logic (counter(3 DOWNT0 0) =
20 "0000"))
#
# But I have not found an easy way to do so. To work around this, I
# declare a tmp signal tmp_counter_ne_zero. (I make sure this signal
# isnt already used by someone else first.) It gets assigned to using
25 # the vhd1 WHEN, ELSE statement:
#         SIGNAL tmp_counter_ne_zero: STD_LOGIC_VECTOR (0 DOWNT0 0) ;
#         tmp_counter_ne_zero <= "1" WHEN (counter(3 DOWNT0 0) = "0000") ELSE
#         "0";
#         --Now we are back in happy std_logic_vector land:
30 #         counter_ne_zero <= NOT (tmp_counter_ne_zero);
#
# All types are converted to STD_LOGIC_VECTOR inside the Entity (VHDL for
# module)
# We convert types to/from STD_LOGIC at the beginning and end of the entity.
35 # e.g.
# module foo(c);
# output c;
# //more verilog code here
# endmodule
40 #
# turns into
# ENTITY foo IS
# PORT (
#     SIGNAL a : IN STD_LOGIC;
45 #     SIGNAL b : INOUT STD_LOGIC;
#     SIGNAL c : OUT STD_LOGIC;
# );
# END foo;
# ARCHITECTURE behavior OF foo IS
50 #     SIGNAL tmp_a : STD_LOGIC_VECTOR(0 DOWNT0 0);
#     SIGNAL tmp_c : STD_LOGIC_VECTOR(0 DOWNT0 0);
#     --INOUT signals are kept as STD_LOGIC_VECTOR
#     --other signals here
# BEGIN
55 #     --verilog code translates to vhd1 here
#     c <= tmp_c(0);
#     tmp_a(0) <= a;
# END behavior
#
60 # IMPORTANT IMPORTANT
# INOUT STD_LOGIC_VECTOR (0 DOWNT0 0) signals are not converted to STD_LOGIC
# INOUT signals are not so easy to convert to STD_LOGIC
# sometimes the INOUT signal looks like an output

```

```

# sometimes its an input. Eventually, I could declare two tmp_signals.
# Convert the signal on the rhs of all equations to be the input version
# signal and convert all signals on the lhs to be the output signal.
# Determine what signal is driving. Then say:
5 # inout_signal = (Driving) ? tmp_output_inout: 1'bz;
# tmp_input_inout = inout_signal.
# For now you'll just have to deal with a STD_LOGIC(0 DOWNT0 0) INOUT.
# Everything instantiating the module will hook up to it correctly.
# END IMPORTANT IMPORTANT
10 #
# V2VHD_Equation can recursively and iteratively call itself.
# The main idea is that complicated expressions
# can be turned into simpler expressions through replacing
# expressions with exclusive words. (see subroutines Replace and
15 Replace_Equivalences)
# Converting an expression with V2VHD_Equation simplifies th
# e.g.
# verilog: c <= a + b + |c;
# Gets turned into
20 # 1) c <= a + b + reduction; // $Equivalence_List(reduction) =
# (c(msb) or c (msb - 1) ... or c(lsb))" // $Width_List(reduction) = 1; #as if
# reduction were a verilog signal.
# 2) c <= addition + reduction // $Equivalence_List(addition) = "a + b";
25 # // $Width_List(reduction) =
# max(width(a),width(b)) + 1;
# 3) c <= tmp_addition // $Equivalence_List(tmp_addition) = addition
# + reduction; // $Width_List(reduction) =
30 # max(width(addition),width(reduction)) + 1;
#
# V2VHD would then return c <= tmp_addition

# V2VHD_Equation_Wrapper is the top level call to V2VHD_Equation. calls
35 &Replace_Equivalences after
# V2VHD_Equation. Replace_Equivalences will replace every
key(%Equivalence_List) with $Equivalence_List{$key}.
# Using c <= tmp_addition from the example before. Replace_Equivalences will
do:
40 # 0) c <= tmp_addition
# 1) c <= addition + reduction;
# 2) c <= a + b + reduction;
# 3) c <= a + b + (c(msb) or c (msb - 1) ... or c(lsb)); --voila, a valid vhd1
equation
45
sub V2VHD_Equation
{
my ($equation,
@Module_Info) = @_ ;
50
my ($Width_List,
$Signal_List,
$Wire_Assignments,
$Equivalence_List) = @Module_Info;
55
my $width;
my $left;
my $right;

60 my $debug = "verilog";

while ($equation =~ s/\s+//sg){};
#Convert all verilog numbers to equivalent word name.

```

```

while ($equation =~ s/(.*?)(\d+\'[bodh](\da-fxz)+)(.*)/$1/si)
{
    my ($replace_this) = &VN2BS($2);
    $width = &Width_Of($2,$Width_List);
5    $replacement_name = &Replace ("Verilog_Number =
$replace_this",$width,@Module_Info);

    $equation .= " $replacement_name $3";
    warn " verilog number equation now is $equation\n"
10    if ($debug =~ /number/);
}

#Convert vector [max_index:min_index] to equivalent word name.
while ($equation =~ s/^(.*?\b)(\w+)\s*(\[.*?\])(.*)/$1/s)
15 {
    my $replace_this = $2;;
    ($replace_this,$left,$right) = &Vector_Range("$2$3",$Width_List);
    my $rest = $5;
    #Worry about memories
    my ($l,$r,$up,$down) = split (/s*\s*\s*/s,$Width_List{$2});
20    if ($up ne "")
    {
        #It's a memory
        #warn "equation is $equation\n";
        warn "\nVERILOG TO VHDL CONVERSION: ".$date_time." USING MEMORY
25 $2\n";
        $replace_this .= "(CONV_INTEGER(unsigned($4)))";
        ($left,$right) = ($l,$r);
    }
    else
30 {
        $right = $left if ($right eq ""); #If only one value,
        #$replace_this .= &Vector_Order($left,$right);
        $replace_this .= "[$left:$right]"; #we'll change vector order when we
35 #replace_equivalences
    }
    #warn "in brackets, (@Module_Info)\n";
    $replacement_name = &Replace("Verilog_Bracket =
$replace_this",$left,$right,@Module_Info);
40 $equation .= "$replacement_name $rest";
}

#WRONG WRONG WRONG WRONG WRONG
#THE ORDER BELOW IS WRONG
45 # Now the order of operators we have (according to verilog 1364 and Samir
Palnitkar's book)
# to handle are
# 1) Parentheses, Replecation and Concatenation
# 2) Unary (+,-,!,~)
50 # 3) Multiply,Divide,Modulus (*,/,% )
# 4) Add, Subtract, Shift (+,-), <<, >>
# 5) Relational (<, <=, >, >=)
# 6) Equality (==,!=)
# 7) Binary Bitwise (&^,|)
55 # 8) Logical (&&,||)
# 7) Reduction (&^,|)
# 9) Conditional (?:)
#10) Assignment (=)

60 #WRONG WRONG WRONG WRONG WRONG
#THE ABOVE ORDER IS WRONG
#AND IS A BUG IN THE 1364 spec.

```

09360106-061201

#SEE THE FOLLOWING POST FROM ALT.COMP.LANG.VERILOG

#...Based on the information obtained from "private sources" (a member
#of the 1364 WG) I concluded that the Table 4-4 is, indeed,
#wrong, and, maybe, the table in Thomas and Moorby 2nd is not
#right, either.
#ALL unary operators have precedence higher than ANY binary operator,
#while the precedence among unary operators is non-essential.
#So, Table 4-4 gives erroneous precedence for ~& and ~| UNARY
#operators placing them where such BINARY operators should have
#been had they existed. It is still to be determined whether
#binary ^~/^ has the same precedence as & (T&M) or lower (IEEE
#Draft), and that can be easily accomplished with Verilog-XL
#which I'm going to do shortly.

#Regards to all,
#Sergei Sokolov

```
my $unary_operators = '\+|\-|!|\~';  
my $reduction_operators =  
'\&{1}|\~\&{1}|\^ {1}|\~\^ {1}|\^~{1}|\|{1}|\~\|{1}';  
my $all_unary_operators = join ('|', $unary_operators, $reduction_operators);  
my $relational_equality_operators = '\<|=|\<|\>|=|\>|\={2}|\!|=';  
my $arithmatic_operators = '\*|\/|\+|\-';  
my $binary_bitwise = '\&{1}|\^ {1}|\|{1}|\~\^|\^~';  
my $shift_operators = '\<<|\>\>';  
  
#It's a pain to differentiate between \&& and \&, so change  
#logical operators to something easier to discern. The ` marks  
#are the verilog escape character, so we know some verilog typer  
#won't accidentally type `AND` accidentally.  
while ($equation =~ s/\&{2}/\`AND\`/){}  
# Similar argument for ||  
while ($equation =~ s/\|{2}/\`OR\`/){}  
my $logical_operators = '\`AND\`|\`OR\`';  
  
#I also hate those === and !== operators which (for all synthesizable  
#code does just the same thing as == and !=  
while ($equation =~ s/\!={2}/\!/=/){}  
while ($equation =~ s/\={3}/\=\/){}  
  
my $binary_operators_with_same_width_operands_and_shift_operators = join  
( '|',  
$arithmatic_operators,  
$binary_bitwise,  
$shift_operators,  
$relational_equality_operators,  
$logical_operators  
);  
  
my $operator_order = join ('|', $all_unary_operators,  
$binary_operators_with_same_width_operands_and_shift_operators,  
$logical_operators);  
  
# 1a) PARENTHESES  
{  
my ($before_paren, $inside_paren, $after_paren) =  
&Count_Parentheses($equation);
```

09830106-0612001
T02T90"90T08860


```

while ($inside_paren ne "")
{
    #####
    # if there is one or more operators inside the parentheses, then
5   process it(them).
    # Otherwise strip the parentheses and continue.
    # i.e. (counter - 1), process it.
    #      (counter)      , return counter
    if ($inside_paren =~ /$operator_order/)
10   {
        #warn "replace_this found ($inside_paren) in ($equation)\n";
        #Recurse equation (DAMN DAMN equation!);
        my $replace_this = &V2VHD_Equation($inside_paren, @Module_Info);
        #warn "Parentheses converted ($inside_paren) to ($replace_this),
15   ($$Width_List{$replace_this})\n";
        die "ERROR V2VHD_Equation: replace_this returned ($replace_this)\n"
            unless ($replace_this =~ s/^\s*(\w+)\s*$/$1/s);
        my $replacement_name = &Replace("Parentheses = $replace_this",
            $$Width_List{$replace_this},
20   @Module_Info);
        $equation = "$before_paren $replacement_name $after_paren";
        #warn "equation now is $equation\n";
    }
    else
25   {
        #warn " parentheses, no operator found\n";
        #put parentheses around
        $equation = "$before_paren $inside_paren $after_paren";
30   ($before_paren,$inside_paren,$after_paren) =
        &Count_Parentheses($equation);
        #warn " parentheses equation now is $equation, width is $width\n";
    }
}
35   # 1b) REPLECATION AND CONCATENATION
    {
        #Replicate first
        #warn " repletion equation was $equation\n";
        while ($equation =~ s/^(.*)\{\s*(\d+)\s*(\{.*\})/$1/s)
40   {
            my $before_curly_brace = $1;
            my $repeat_number = $2;
            my ($b,$m,$e) = &Count_Parentheses($3,\s*\{','\}\s*');
45   warn "repletion and concatenation bme is($b),($m),($e)\n"
                if ($debug =~ /repletion/i);
            $m = ("m\," x $repeat_number);
            $m =~ s/\\,\s*$//s; #lose last comma.
            $e =~ s/^\s*\}\\//s;
50   $equation = "$before_curly_brace $b\{$m\}$e";
            #warn " repletion equation now is $equation\n";
        }

        #All other curly_braces ({,}) (including the repletion we handled
55   above) are concatenation
        my ($before_curly_brace,$inside_curly_brace,$after_curly_brace) =
        &Count_Parentheses($equation,\{\{','\}\});
        while ($inside_curly_brace ne "")
60   {
            #warn "BPC ($inside_curly_brace) \n";
            #warn %$Width_List;
            my $replacement_name = &Process_Concatenation
            ("\"{$inside_curly_brace}\"",@Module_Info);

```

```

$equation = "$before_curly_brace $replacement_name
$after_curly_brace";
($before_curly_brace,$inside_curly_brace,$after_curly_brace) =
&Count_Parentheses($equation,'\{','\}');
5   }
   }

# 2) UNARY OPERATORS
{
10  #####
   # In addition to the normal unary operators
   # Verilog has these weird but useful unary operators called reduction
operators
   # a = |foo[3:0] => a = foo[3] | foo[2] | foo[1] | foo[0];
15  # It turns out that these operators are quite useful.
   # However it is difficult to distinguish between the unary reduction
operator
   # and the binary "or" operator. a = c | foo or a = c + |foo looks quite
like a = |foo.
20  #
   # Here is how we differentiate. If the first non spaced character before
the possible
   # reduction character is a word (\w) then consider it a binary operator.
If it is a
25  # non word character, consider it a unary operator. This is good even
for equations with
   # parentheses because we've already converted parentheses to words.
   # Consider: a = (c + b) | foo
   # Above, we've already converted (c + b) to tmp_parentheses so the
30  equation we see is
   # a = tmp_parentheses | foo //binary or
   #

while ($equation =~ /($all_unary_operators)/)
35  {
   my $operator = &Find_In_Order($equation,$all_unary_operators);
                           last unless ($equation =~
s/^(.*?[^\\w\\s])\\s*($all_unary_operators)\\s*(\\w+)\\s*(.*)/$1/s ||
                           $equation =~
40  s/^(\\s*)($all_unary_operators)\\s*(\\w+)\\s*(.*)/$1/s);
   my $beginning = $1;
   $operator = $2;
   my $name = $3;
   $width = &Width_Of($name,$Width_List);
45  my $rest = $4;
   my $replace_this;
   if ($operator =~ /^$unary_operators$/)
   {
       if ($operator eq "\\!")
50       {
           if ($width == 1){$replace_this = " (NOT $name)";}
           else{$replace_this = "\\($name \\= \\\".\"(\"0\" x $width).\"\\\"\\)";}
           $width = 1;
       }
       else
       {
           $operator =~ s/~/NOT/;
           $replace_this = "($operator $name)";
       }
60  }
   else
   {
       my $value_to_reduce = $name;

```

09880106 "061201

```
my $reduction_operator = $operator;
```

```
$reduction_operator = "OR" if ($reduction_operator =~ /\^|\{1}$/);
```

```
$reduction_operator = "AND" if ($reduction_operator =~ /\^&\{1}$/);
```

```
5 $reduction_operator = "NOR" if ($reduction_operator =~ /\^~\{1\}\| \{1\}$/);
```

```
$reduction_operator = "NAND" if ($reduction_operator =~ /\^~\{1\}&\{1\}$/);
```

```
$reduction_operator = "XOR" if ($reduction_operator =~ /\^^\{1\}$/);
```

```
10 $reduction_operator = "XNOR" if ($reduction_operator =~ /\^^\{1\}~$/);
```

```
$reduction_operator = "XNOR" if ($reduction_operator =~ /\^~^\{1\}$/);
```

```
15 my ($value_to_reduce, $vec_left, $vec_right) =  
&Vector_Range($value_to_reduce, $Width_List);
```

```
if ($vec_right ne "")
```

```
{
```

```
20 #We can just expand the bit vectors if width is left, right  
because reduction is simple i.e. |a
```

```
($replace_this, $tmp) =
```

```
&Expand_Array_Of_Bit_Vectors_Into_Separated_Bits(" $reduction_operator",
```

```
$value_to_reduce,
```

```
25 @Module_Info);
```

```
}  
else
```

```
{
```

```
30 #else it's not so simple, so generate a signal = to rhs and  
then reduce the signal
```

```
#warn "reduction operator $reduction_operator, vtr $vtr,  
$tmp_width\n";
```

```
#We need to first make an intermediate signal since there is an  
operator inside
```

```
35 my ($new_signal_name) = &Add_Intermediate_Signal  
("reduced_$reduction_operator = $value_to_reduce",
```

```
&Width_Of($value_to_reduce, $Width_List),
```

```
@Module_Info);
```

```
40 ($replace_this, $tmp) =
```

```
&Expand_Array_Of_Bit_Vectors_Into_Separated_Bits(" $reduction_operator",
```

```
$new_signal_name,
```

```
45 @Module_Info);
```

```
}  
#Turn indexed bits into bit array of size 1
```

```
while ($replace_this =~ s/\[\s*(\d+)\s*\]/\[$1:$1\]/s){}
```

```
50 $width = 1;  
$replace_this = "($replace_this)";
```

```
}
```

```
my $replacement_name = &Replace ("unary = $replace_this",  
$width,
```

```
@Module_Info);
```

```
$equation .= "$replacement_name $rest";
```

```
}
```

```
60 }
```

```
# 3,4, BINARY OPERATORS
```

```
# 5,6 RELATIONAL (<, <=, >, >=) AND EQUALITY (==, !=)
```

```
# AND LOGICAL
```

09880106.061201

```

{
    #warn " pre-relational equation is $equation\n";
    #die "bowslop" is
    $binary_operators_with_same_width_operands_and_shift_operators\n";
    while ($equation =~
5      /($binary_operators_with_same_width_operands_and_shift_operators)/s)
    {
        my $operator =
10      &Find_In_Order($equation,$binary_operators_with_same_width_operands_and_shift_o
perators);
        die "BINARY_OPERATOR ($operator) not found in ($equation)\n"
        unless ($equation =~ s/^(.*?)(\w+)\s*($operator)\s*(\w+)(.*)/$1/s);
        my $left_operand = $2;
        $operator = $3;
15      #warn "operator now is $operator\n";
        my $right_operand = $4;
        my $rest = $5;
        my $replace_this;
        if (&Is_real($left_operand) && &Is_real($right_operand))
20      {
            #warn " relational equation is real\n";
            #If both are real numbers, evaluate using perl's
            #binary operators.
            if ($operator eq "\`AND\`")
25      {
                if (($2 == 0) || ($4 == 0)){$equation .= "0";}
                else{$equation .= "1";}
            }
            else
30      {
                if ($operator eq "\`OR\`")
                {
                    if (($2 != 0) || ($4 != 1)){$equation .= "1";}
                    else {$equation .= "0";}
                }
                else
35      {
                    my $evaluatedExpression = eval ("$left_operand $operator
$right_operand");
40
                    # Programming Perl p. 87:
                    # 'The equal and not-equal operators return 1 for true, and
                    "" for false
                    # just as the relational operators do).'
45      # Translate all forms of "false" to 0:
                    $evaluatedExpression = "0" if (!$evaluatedExpression);

                    $equation .= $evaluatedExpression;
                }
50      }
            $equation .= $rest;
            next;
        }

55      #shift_operators operate on integers, all other operators need their
integers to be converted to bit_vectors.
        if ($operator =~ /^$shift_operators$/)
        {
            if (&Is_real($left_operand))
60      {
                $replacement_name = eval "$left_operand $operator
$right_operand";
                # Translate all forms of "false" to 0:
            }
        }
    }
}

```

09880106_061201
102190" 90108860

```

        $replacement_name = "0" if (!$replacement_name);
    }
    else
    {
5         my ($name,$left,$right) =
        &Vector_Range($left_operand,$Width_List);
        $width = &Width_Of($left_operand,$Width_List);
        die "ERROR V2VHD_EQUATION ($equation) HAS UNKNOWN width. UNABLE
10     TO SHIFT\n"
        if ($left eq "");

        die "ERROR V2VHD_EQUATION ($equation) HAS NON-INTEGGER SHIFT
15     AMOUNT\n"
        unless ($right_operand =~ s/^\s*(\d+)\s*$/$1/s);

        if ($right eq "")
        {
            $name = &Add_Intermediate_Signal ("shift = $left_operand",
20                $left_operand_left_index,
                @Module_Info
                );

            $left--;
            $right = .0;
        }

25     #####
        #c [3:0] >> 1 = c [3:1];
        #c [0:3] >> 1 = c [0:2];

30     my @array = &Order($left,$right);
        my $zeros;
        while ( ($right_operand--) && ($width != 0) )
        {
            if ($operator eq "<<<"){$zeros .= "0"; $width++;}
35         else{
                if ($operator eq ">>>"){pop(@array);$width--}
                else {die "ERROR V2VHD_EQUATION, UNKNOWN SHIFT
40     ASSIGNMENT($operator)\n";}
            }
        }

        my $expand_this_string;
        if (@array)
        {
45             $left = $array[0];
            $right = $array[-1]; #last value in the array
            $expand_this_string = "$name\[ $left\:$right\]";
            if ($zeros ne "")
            {
50                 $expand_this_string .= ", \"$zeros\"";
            }
            #warn "sending $expand_this_string to EAOBVISB\n";
            ($replace_this,$width)
            =
55     &Expand_Array_Of_Bit_Vectors_Into_Separated_Bits ("", "",
            $expand_this_string,
            @Module_Info
            );
60     $replace_this = "std_logic_vector' ( $replace_this)\n";
        }
    else

```

```

5      {
        $replacement_name = "0";
      }

    }
    my $replacement_name = &Replace("shift" =
$replace_this",$width,@Module_Info);
    $equation .= "$replacement_name $rest";
    next;
10  }

    if ($operator =~ /^$logical_operators$/)
    {
        #operator transforms
        #warn "found lo ($left_operand,$operator,$right_operand)\n";
        $operator = "AND" if ($operator eq "\\`AND\\`");
        $operator = "OR" if ($operator eq "\\`OR\\`");
        $replace_this =
20  "(. &Process_If_Condition($left_operand,@Module_Info).) $operator ("
        .&Process_If_Condition($right_operand,@Module_Info).)";
        $width = 1;

        my $replacement_name = &Add_Intermediate_Signal ("logical = \\`1\\`"
WHEN ($replace_this) ELSE \\`0\\`",
        $width,
        @Module_Info);

        $equation .= "$replacement_name $rest";
        next;
30  }

        #warn "calling CV2SLV ($left_operand,$operator,$right_operand)\n";
        ($width,$left_operand,$right_operand) =
&Convert_Integers_To_Std_Logic_Vector($left_operand,
        $operator,
35  $right_operand,
        $width_List,
        $Equivalence_List);
        if ($operator =~ /^$arithmetic_operators$/)
        {
            # VHDL thinks c = a + b results in c_width = max(a_width,b_width).
            # But that is clearly wrong. In reality, c_width =
45  max(a_width,b_width) + 1.
            # So we use reconcile widths to give us signals (new_a,new_b) that
            have widths (a_width+1,b_width+1);
            # We use reconcile known widths to get us a signal of the correct
50  width.
            my $new_width;
            if ($operator =~ /(\\`+|\\`-)/)
            {
                $new_width = $width + 1; #Max_width from
55  Convert_Integers_To_Std_Logic_Vector above.
                $left_operand = &Reconcile_Known_Widths(
                    $left_operand,
                    $left_operand,
                    $new_width,
60  &Width_Of($left_operand,$width_List),
                    @Module_Info
                );
            }
        }
    }
}

```

```

    $right_operand = &Reconcile_Known_Widths(
                                                $right_operand,
                                                $right_operand,
                                                $new_width,
5
        &Width_Of($right_operand,$Width_List),
                                                @Module_Info
                                                );

10
    }

    if ($operator == /\*/)
    {
        $new_width = &Width_Of($left_operand,$Width_List) +
15 &Width_Of($right_operand,$Width_List) ;
    }
    if ($operator == /\//)
    {
        $new_width = &Width_Of($left_operand,$Width_List) -
20 &Width_Of($right_operand,$Width_List);
    }
    #warn "worop is ".$&Width_Of($right_operand,$Width_List).", nw is
    $new_width\n";

25
    $width = $new_width;
    #warn "width is ($width)\n";
    $replace_this = "$new_left_operand $operator $new_right_operand";
    $replace_this = "$left_operand $operator $right_operand";
    my $replacement_name = &Replace("arithmetic" =
30 $replace_this",$width,@Module_Info);
    $equation .= "$replacement_name $rest";
    next;
}

35
#relational operator transforms
if ($operator == /^$relational_equality_operators$/)
{
    #####
    # Convert boolean (in)equality. Since the output of a conditional
40 is a boolean
    # and we only understands std_logic, we need to replace the
    conditional with
    # a wire named something like tmp_std_logic.
    # Then we set tmp_std_logic <= '1' when (condition) else '0';
45
    #
    # N.B. In the future, since we convert everything to
    std_logic_vector, we might
    # be able to convert the boolean to std_logic_vector

50
    my $tmp_left =
    &Replace_Equivalences($left_operand,$Equivalence_List);
    my $tmp_right =
    &Replace_Equivalences($right_operand,$Equivalence_List);

55
    my $tmp_output_name;
    #warn "operator ($operator) is relational_operator\n";
    if ($operator == s/\s*\={2}\s*/ \= /s) {$tmp_output_name =
    "$tmp_left\_eq\_tmp_right";}
    if ($operator == s|\s*\!\={1,2}\s*| \/\= |s) {$tmp_output_name =
60 "$tmp_left\_ne\_tmp_right";}
    if ($operator == /\s*\<\s*/s) {$tmp_output_name =
    "$tmp_left\_lt\_tmp_right";}

```

```

    if ($operator =~ /\s*<=\s*/s)                                { $tmp_output_name =
"$tmp_left\_le\_tmp_right"; }
    if ($operator =~ /\s*>\s*/s)                                { $tmp_output_name =
"$tmp_left\_gt\_tmp_right"; }
5    if ($operator =~ /\s*>=\s*/s)                                { $tmp_output_name =
"$tmp_left\_ge\_tmp_right"; }

    if ($tmp_output_name)
    {
10        my $boolean = "$left_operand $operator $right_operand";
        $tmp_output_name = &Add_Intermediate_Signal("$tmp_output_name =
\"1\" WHEN \"($boolean)\" ELSE \"0\"",
                                                    1,
                                                    @Module_Info);

15        $equation .= "$tmp_output_name$rest";
        #warn "ton, ($equation)\n";
    }
    next;
20 }

    $operator = "XOR" if ($operator =~ /\^\^$/);
    $operator = "AND" if ($operator =~ /\^\&$/);
    $operator = "OR" if ($operator =~ /\^\|$/);
25    $operator = "XNOR" if ($operator =~ /\^\^~$/);
    $operator = "XNOR" if ($operator =~ /\^\~\^$/);

    my $replacement_name = &Replace("binary_operator = $left_operand
$operator $right_operand",
30                                $width,
                                @Module_Info);
    $equation .= "$replacement_name$rest";
}
}

35 #CONDITIONAL ?:
    while ($equation =~ s/(\s*)(\w+)\s*?\s*(\w+)\s*?: (\s*)/"$1$3 WHEN
\("&Process_If_Condition($2,@Module_Info)."\) ELSE $4"/sge)
    { #warn "found conditional($2)\n"
40     ; }

    # ASSIGNMENT
    # The following is copied very closely from the
    $binary_operators_with_same_width_operands case above
45    # Assignment gets a little bit nasty because unlike verilog, VHDL requires
    that the left and right sides of
    # assignments be the same width. So we need to chop off or add bits to the
    right hand side of the assignment.
    # We do that in &Reconcile_Widths.
50    {
        my $tmp_equation = $equation;
        if ($equation =~ s/^(.*) (\w+)\s*(\={1})\s*(\w+) (.*)/"$1$2 $3/s)
        {
55            my $left_operand = $2;
            my $operator = $3;
            my $right_operand = $4;
            my $rest = $5;

            ($width,$left_operand,$right_operand) =
&Convert_Integers_To_Std_Logic_Vector($left_operand,
60                                     $operator,
                                     $right_operand,
```



```

$Width_List,
$Equivalence_List);
5
    #warn "    left_operand, operator, right_operand, rest, width,
replace_this IS $left_operand, $operator, $right_operand, $rest, $width,
$replace_this\n";
    $right_operand = &Reconcile_Widths($left_operand,
10
                                $right_operand,
                                @Module_Info
                                );
    #warn "    ro is $right_operand\n";
    #operator transforms
15
    #replace_this = "$operator $right_operand";
    $equation .= "$right_operand$rest"; #replace_this$rest";
    #warn "    assignment equation now is $equation\n";
    )

20
    #We also need to match up lhs with values following ELSE statements
    #I put a `` mark before all processed ELSE words to keep us from an
infinite loop.
    #Again, we need to Reconcile_Widths.
    while ($equation ==~
25
s/^(.*) (\w+) (\s*=\{1\}.*?ELSE(\s+|[\^\\`w])) (\w+) (.*)/$1/s)
    {
        my $left_operand = $2;
        my $in_between = $3;
        my $right_operand = $5;
30
        my $rest = $6;
        ($width,$left_operand,$right_operand) =
&Convert_Integers_To_Std_Logic_Vector($left_operand,
                                "=",
35
    $right_operand,
    $Width_List,
    $Equivalence_List);
40
    #warn "    Conditional Assignment: left_operand, in_between,
right_operand, rest, width, replace_this IS $left_operand, $in_between,
$right_operand, $rest, $width, $replace_this\n";

    #Do the same width reconciliation deal as above. (if needed)
45
    $right_operand = &Reconcile_Widths($left_operand,
                                $right_operand,
                                @Module_Info
                                );

50
    #in_between transforms
    $replace_this = "$left_operand $in_between \\\`$right_operand";
    $equation .= "$replace_this$rest";
    #warn "    ELSE assignment equation now is $equation\n";
    )

55
    #crush `marks
    while ($equation =~ s/\\`//g){;}
    }

60
    $equation =~ s/^\s*(.*)\s*$/1/s;

    return($equation);
}

```



```

    );
}

sub Reconcile_Known_Widths
5  {
    my ($left_operand,
        $right_operand,
        $left_width,
        $right_width,
10     @Module_Info) = @_;

    my ($Width_List,
        $Signal_List,
        $pWire_Assignments,
15     $Equivalence_List) = @Module_Info;

    #warn ("RNW, $left_operand,
    # $right_operand,
    # $left_width,
20     # $right_width\n");

    #We may not know the left or right operand on pass 1,
    #die "Reconcile_Widths, left_width not known" if ($left_width eq "");
    #die "Reconcile_Widths, right_width not known" if ($right_width eq "");
25     return ($right_operand) if ($left_width eq "");
    return ($right_operand) if ($right_width eq "");

    my $tmp_signal;
    if ($left_width == $right_width)
30     {
        #warn "$left_operand has same width as $right_operand, returning\n";
        return ($right_operand);
    }
    else
35     {
        $tmp_signal = &Add_Intermediate_Signal
        (" $left_operand\_ $right_width\_bits\_wide = $right_operand",
         $right_width,
         @Module_Info);
40     }

    if ( $left_width < $right_width)
    {
        #warn " RW: lw < rw.\n";
45     $right_operand = "$tmp_signal (($left_width - 1) DOWNT0 0)";
    }

    if ( $left_width > $right_width)
    {
50     my $width_difference = $left_width - $right_width;
        my $filler = "\'0\'", " x $width_difference;
        my ($EAOBV,$junk) =
        &Expand_Array_Of_Bit_Vectors_Into_Separated_Bits("",$tmp_signal,@Module_Info);
        $right_operand = "std_logic_vector\' ($filler$EAOBV)";
55     }
    return ($right_operand);
}

#####
60 #
# Add_Intermediate_Signal takes a vhdl string signal "a" or "a = b" and
# returns
# the correct signal name to use. If signal is of the form "a = b" it replaces

```

equivalences of b and adds a <= b to wire_assignments

sub Add_Intermediate_Signal
{

5 my (\$signal,
 \$width,
 @Module_Info) = @_;

10 my (
 \$Width_List,
 \$Signal_List,
 \$Wire_Assignments,
 \$Equivalence_List) = @Module_Info;

15 my (\$signal_lhs, \$signal_rhs) = split (/^s*={1}s*/s, \$signal, 2);
 #my \$signal_rhs = join ("=", @signal_rhs);
 \$signal_rhs = &Replace_Equivalences(\$signal_rhs, \$Equivalence_List)
 if (\$Equivalence_List && \$signal_rhs);

20 die "ERROR Add_Intermediate_Signal: ILLEGAL WIDTH (\$width) IN SIGNAL
ASSIGNMENT (\$signal)\n"
 unless ((\$width =~ s/^s*(\d+)\s*/\$1/) && (\$width > 0));

25 #just return signal_rhs if only one value is on the rhs.
 #eg. foo = bar, just return bar
 if (\$signal_rhs =~ s/^s*(\w+)\s*/\$1/s)

30 {
 #Orion, do width checking here
 return (\$signal_rhs);
 }

35 #rename signal_lhs
 \$signal_lhs = &Get_Exclusive_VHDL_Name(\$signal_lhs, \$Width_List);
 if (\$signal_rhs ne "")
 {
 \$\$Wire_Assignments .= " \$signal_lhs <= \$signal_rhs;\n";
 }

40 \$\$Signal_List{\$signal_lhs} = "SIGNAL \$signal_lhs :
STD_LOGIC_VECTOR(\".(\$width - 1).\" DOWNT0 0);";
 \$\$Width_List{\$signal_lhs} = (\$width-1).",0";
 return (\$signal_lhs);
}

45 #####
 #
 # Is_real returns 1 if the value passed to it is a real number, i.e integer >=
 0
 #####

50 #
 sub Is_real
 {
 my \$value = shift (@_);
 return (1) if (\$value =~ /^s*\d+\s*/s);
55 return (0);
 }

60 #####
 #
 # Count_Parentheses
 #
 # so i have a string always @(blow(me)(leonardo|synplicity)) blerg (boof)
 # I want to perform computations on the string surrounded by the

09880106.061201

```

# beginning and last parentheses. I call Count_Parentheses and it
# returns 3 values, the beginning string: "always @", the parenthesized string
# "blow(me) (leonardo|synplicity)" and the last string "blerg (boof)".
#
5 # If I want to search on something other than parentheses, say begin,end, I can
# place their values in $begin_match and $end_match.
#####
#
10 sub Count_Parentheses
{
    my ($string,$begin_match,$end_match) = @_ ;
    my $begin_string;
    my $paren_string;
    my $end_string;
15     my $begin_match_default = '\s*\(\s*';
    my $end_match_default = '\s*\)\s*';
    $begin_match = $begin_match_default unless ($begin_match);
    $end_match = $end_match_default unless ($end_match);

20     warn "          CP string is $string,\n"
        ." $begin_match,\n $end_match\n"
        if (0);
    #if ($begin_match ne $begin_match_default);

25     return("", "", "$string")
        unless ($string =~ /^(.*?)$begin_match(.*)$/s);

    #warn "          found first bit $1, \n"
    rest is $2\n    in string $string\n"
30     #if ($GLOBAL_WARNING);
    # ." $begin_match,\n $end_match\n"
    #if ($begin_match ne $begin_match_default);

    $begin_string = $1;
35     my $paren_count = 1;
    $end_string = $2;

    while ($end_string =~ s/^(.*?)($begin_match|$end_match)(.*)$/3/s)
40     {
        my $match;
        $match = $2;
        $paren_string .= $1;

45         if ($match =~ /$begin_match/)
        {
            $paren_count++;
        }
        else
50         {
            $paren_count = $paren_count - 1;
        }

        last if ($paren_count == 0);
55         #else
        $paren_string .= $match;
    }

    #die "mismatched $begin_match,$end_match in string"
60 $begin_string$paren_string$end_string" if ($paren_count != 0);
    return ($begin_string,$paren_string,$end_string);
}

```

09880106 061201

sub Handle_Next_If_Else_Line

```
{
    my ($if_body, @Module_Info) = (@_);

5    my (
        $Width_List,
        $Signal_List,
        $pWire_Assignments,
        $Equivalence_List) = @Module_Info;

10    my $rest_of_module;

    my $spacing;
    #####
15    # following a verilog "if(condition)", "always @(condition), else,
statement, there are three things that
    # can follow the statements.
    # 1) an if statement
    # 2) a begin - end block
20    # 3) a one line statement ending with a semi-colon;

    # 1) an if statement
    if ($if_body =~ /^(\s*)\bif\s*(\s*)/s)
    {
25        my ($tmp, $if_conditions, $tmp_if_body) = &Count_Parentheses($2);
        #warn "if statement is ($if_conditions)\n";
        $if_body = "$1IF " . &Process_If_Condition($if_conditions, @Module_Info) . "
THEN ";
        ($tmp_if_body, $rest_of_module) =
30        &Handle_Next_If_Else_Line($tmp_if_body, @Module_Info);
        $if_body .= $tmp_if_body;

        #If an else statement follows the block
        #Process it.
35        if ($rest_of_module =~ /^(\s*)else\s*(\s*)/s)
        {
            $if_body .= "$1ELSE";
            ($tmp_if_body, $rest_of_module) =
40            &Handle_Next_If_Else_Line($2, @Module_Info);
            $if_body .= $tmp_if_body;
        }
        $if_body .= "\n$spacing"."END IF;";
        return ($if_body, $rest_of_module);
    }

45    # 2) a begin - end block
    if ($if_body =~ /^(\s*)\bbegin\b/s)
    {
50        #warn "hniel, begin end line, ib is $if_body\n";
        #if ($HNIEL_DEBUG);
        my ($tmp, $tmp_body, $rest_of_module) =
        &Count_Parentheses($if_body, '\bbegin\b', '\bend\b');

        #if we are in a begin_end block, then process each command. If there is
55        an if statement
        #lurking, then call this function again.

        #my @commands = split (/\/s, $tmp_if_body);
        my $line;
        $if_body = "";
60        while (!($tmp_body =~ /^(\s*)$/s))
        {
            ($tmp, $tmp_body) = &Handle_Next_If_Else_Line($tmp_body,
```

09830106-061201

@Module_Info),

```

    $if_body .= $tmp;
    #warn "begin_end, ib($if_body)\n--tmp_body($tmp_body)\n--\n"
    #if ($HNIEL_DEBUG);
5      )
    #warn "hniel, ib now is ($if_body)\n"
    #if ($HNIEL_DEBUG);
    return ($if_body,$rest_of_module);
10  }

# 3) a one line statement ending with a semi-colon;
if ($if_body =~ /^(\s*)(.*?;)(.*)/s)
{
    #warn "hniel, oneline ($2)\n"
    #if ($HNIEL_DEBUG);
15    $if_body = $1.&Process_Register_Assignment($2,@Module_Info);
    #warn "          becomes ($if_body)\n"
    #if ($HNIEL_DEBUG);

20    $rest_of_module = $3;
    return ($if_body,$rest_of_module);
}

}

25 #####
#
# Process_If_Condition: converts verilog if conditions
# to VHDL IF conditions.
# V2VHD_Equation
30 #####
#
sub Process_If_Condition
{
    my ($condition, @Module_Info) = (@_);
35    my (
        $Width_List,
        $Signal_List,
        $pWire_Assignments,
        $Equivalence_List) = @Module_Info;

40    #warn "PIC, $condition\n";
    my %E_List;
    if ($Equivalence_List eq "")
    {
45        $Equivalence_List = \%E_List;
        $Module_Info[3] = $Equivalence_List;
    }
    #relational_equality_operators, and transforms copied from V2VHD_Equation;
    my $vhdl_relational_equality_operators = '\<|=|\<|\>|=|\>|\={1}|\|/\=';
50    # If there is no relational_equality_operators in the equation, it's
    # implied that the equation != 0. i.e.
    # if (foo) //same thing as if (foo != 0);
    # if (!foo) // same thing as if (!foo != 0);
    my $result = &V2VHD_Equation($condition,@Module_Info);

55    my $result_width = &Width_Of($result,$Width_List);
    #die "ERROR Process_If_Condition, WIDTH FOR ($result) NOT KNOWN in
    ($condition)!\n"
    #unless ($result_width);
    my $rhs = 0 x $result_width;

60    $result = &Replace_Equivalences($result,$Equivalence_List);
    my $return_string = "$result";
}
```

09890106-061201

```

if ($result != /$vhdl_relational_equality_operators/)
{
    if ($result =~ /\s*\("[01]+\s*\s*$/)
    {
        if ($result =~ /1/)
        {
            $return_string = "true";
        }
        else
        {
            $return_string = "false";
        }
    }
    else
    {
        $return_string = "($result) /\s= \"$rhs\"";
    }
}
#warn "PIC, returns $return_string\n" if $start_special;
return ($return_string);
}

```

```

#####
#
# Get_Attribute_Types: Returns all attributes defined by VHDL code.
# Currently, I have just pasted in the attribute definitions from exemplar.vhd
# into a "HERE" string;
#####

```

```

sub Get_Attribute_Types

```

```

{
    my ($pAttribute_List) = @_;

    my $vhdl_string = q[
        ---
        -- Attribute declarations
        ---

        attribute required_time      : time ;
        attribute arrival_time       : time ;
        attribute output_load        : real ;
        attribute max_load           : real ;
        attribute clock_cycle        : time ;
        attribute clock_offset       : time ;
        attribute pulse_width        : time ;
        attribute input_drive        : time ;
        attribute nobuff             : boolean ;
        attribute pin_number         : string ;
        attribute preserve_signal    : boolean ;
        attribute noopt              : boolean ;
    ]

```

```

        -- New attributes in 2.1 release

        -- Specify pin_numbers for bits of a 1-dimensional array
        type exemplar_string_array is array (natural range <>,
natural range <>) of character ;
        attribute array_pin_number : exemplar_string_array ;
    ]

```

```

        -- Buffer_sig attribute to force a (clock) buffer on a signal
        attribute buffer_sig : string ;
    ]

```

```

        -- PAD attribute to force a particular PAD cell on a IO pin
    ]
}

```



```

-- Does not work for Xilinx, Orca and Altera.
attribute pad : string ;

```

```

5  generators
    -- Type needed to indicate speed requirements for module
    type modgen_select is (smallest, small, fast, fastest);
    -- Use this attribute to set speed on signals/variables
    attribute modgen_sel : modgen_select ;

10    -- New attributes in 2.2 release
    -- Attributes for encoding of enumerated types.

    type encoding_style is (BINARY, ONEHOT, TWOHOT, GRAY, RANDOM)
15    ;
    attribute TYPE_ENCODING_STYLE : encoding_style ;

    -- Example of using type_encoding_style for an enumerated
20    type :
    --      type state_t is (PLAY, WAIT_FOR_MOVE, END_OF_GAME);
    --      attribute TYPE_ENCODING_STYLE of state_t:type is
    ONEHOT ;

    attribute TYPE_ENCODING : exemplar_string_array ;

25    -- Example of using TYPE_ENCODING for an enumerated type :
    --      type state_t is (PLAY, WAIT_FOR_MOVE, END_OF_GAME);
    --      attribute TYPE_ENCODING of state_t:type is
    ("011", "110", "101") ;
30    ];

    #Crush comments
    while ($vhdl_string =~ s/^(.*?)\-\-\-.*$/\1/m){;}

35    my @commands = split (/\\s*\\;\\s*/s,$vhdl_string);
    foreach $command (@commands)
    {
        if ($command =~ /^\\s*attribute\\s+(\\w+)\\s+\\:\\s*(\\w+)/s)
        {
40            $$pAttribute_List{$1} = $2;
            #warn "Type $1 is $2\\n";
        }
    }

45    }

#####
#
# V2VHD converts from a synthesizable verilog file to a synthesizable vhd1 file
#
50    # Still to be done
    # 1) Make multiple behaviour modules for each definition
    # 5) Save comments around modules and always blocks
    # 6) Convert $display statements
    # 7) Make a special altera_verilog library that overloads verilog operators
55    # That will make for a much cleaner compiler.
    #
#####
#
sub V2VHD
60    {
    my ($Verilog_String,
        $complete_filename,
        $pass,

```

09880106.061201
T02290.90T08901

```

$Module_Indexed_Port_Width_Pointer,
$Module_Indexed_Port_Names_Pointer,
$Module_Indexed_Parameter_List_Pointer,
$Component_List_Pointer) = @_;

```

```

5 my %Entity_And_Architecture;
my %Module_Instantiation_List;
my @Module_Array;

```

```

10 #Put the whole shebang including `included files in $line.
$complete_filename =~ tr|\\|\/|;
my @tmp_path = split (/\/, $complete_filename);
my $filename = pop (@tmp_path);
my $path = join ("\/", @tmp_path);
15 #warn "path,filename $path,$filename\n";
my $line = $Verilog_String;
$line .= &read_file('fh000', $filename, $path);

```

```

20 my $all_entity_declarations;
my $all_architecture_blocks;
my %DefParam_List;
my $vhdl_module = "--$complete_filename\n\n";

```

```

25 #####
# Just like leonardo spectrum, we need to know components.
# unfortunately, we forgot to include ifdef SIMULATION around the
# component declaration. So we hack up a search using commented
# information and stick the module back in $line after line has
# been cleaned up.
30 my $tmp = $line;
my @components = ();
while ($tmp =~
    s|\/\/\/\s+Black\-box\s+declaration\s+for\s+simple\s+module\s+(\w+)\s+
    (module\s+\1.*?)
    35 \\/\*\s+synthesis\s+syn_black_box\s+\*\/
    (.*?)\bendmodule)
    ||sx)
{
    40 push (@components, $2.$3);
}
#Find and process comments
$line = &Kill_Comments($line);
#$line = &Process_Comments($line);

```

```

45 #Find definitions and process ifdefs
$line = &Process_Verilog_Directives($line);

```

```

#stick those components back in.
$line .= join (" ", @components);
50 my %attribute_type;
&Get_Attribute_Types (\%attribute_type);

```

```

#Process each module

```

```

55 while ($line =~ s/^((.*?)\bmodule\s+(\w+)\s+.*?)\bendmodule\b(.*?)$/\1$4/s)
{
    my $module_name;
    $module_name = $2;
    60 #warn "module $module_name, pass $pass\n";
    if ($pass == 2)
    {

```

09880106-061201

```

    push (@Module_Array,$module_name);
}

undef %Parameter_List;
5 my %Parameter_List;
  my @parameter_order = ();
  my $module_commands = $3;
  my $Process_Statements;
  my $Wire_Assignments = "";
10 my %Declared_Components;

  my $instantiation_string;
  my $attribute_string;

15 my %attribute_already_declared;

  #Clean up module_name
  $module_name =~ s/^\s*(.*?)\s$/\1/s;
  #warn "module name is $module_name\n";
  warn "VERILOG TO VHDL CONVERSION: ".&date_time." PROCESSING MODULE
20 $module_name\n";

#####
# We need to know all defparams before we instantiate a module
# So we look for defparams separately and store them in
# %DefParam_List which is indexed by instance name.
# When it comes time to instantiate a module, we will have all
# the information we need.
while ($module_commands =~ s/^(.*?)\bdefparam\s+(.*?)\;(.*)/\$1\$3/s)
30 {
    #defparam instance_name.parameter = value,
    # instance_nameX.parameterY = valueZ ;
    #print "found defparam $2\n";
    foreach $defparam (split (/^\s*\s*\s*/s,$2))
    {
35         if ($defparam =~ /^^\s*(\w+)\.(\w+)\s*\s*=\s*(\S+)/)
        {
            $DefParam_List{$1} .= ",\n" if ($DefParam_List{$1});
            $DefParam_List{$1} .= "    $2 => $3";
            #warn "adding $2 => $3 to dplist $1\n";
40         }
        else
        {
            die "ERROR: defparam statement $defparam not understood!";
45         }
        }
    }
}

#####
50 # We need to find all exemplar attributes separately of the
# commands that are split by semi-colon.
# So we look for exemplar attributes separately and store them
# When it comes time to instantiate a module, we will have
# the attribute information that we need.
55 # warn "before exemplar check, $module_commands\n----\n";

# NO EXEMPLAR ATTRIBUTES ARE RESPECTED, MODULES WITH NO CONTENTS WILL
# BE CONVERTED TO BLACK BOXES
while (0)#{($module_commands =~ s/^(.*?)\b-
60 \s*exemplar\s+attribute\s+(.*?)\s*/\1/mi)
{
    my $attribute_info = $2;
    my ($name,$attribute,$attribute_value) = split (/^\s*/s,$2);

```

09380106_061201
T02T90" 90T08960

```

    $attribute =~ tr/A-Z/a-z/;
    my $att_type = $attribute_type($attribute);
    #warn "attribute found: name,attribute,attribute_value =
$name,$attribute,$attribute_value,$att_type\n";
5      if ($att_type)
        {
            $attribute_string .= "        --attribute $attribute : $att_type;\n"
            unless ($attribute_already_declared{$attribute});
            $attribute_already_declared{$attribute}++;
10      $attribute_string .= "        --attribute $attribute of $name:ENTITY
is $attribute_value;\n";
        }
    }
15    #warn "after exemplar check, $module_commands\n---\n";

    #All other commands are separated by ";" and are processed inline with
each command.

20    #####
    # Port_List only contains information for module ports
    # Signal_List only contains information for wires and registers
    #
    # Port_Width, Port_Type contains info for all ports, wires and registers
    # They should probably be renamed All_Nodes_Width/Type or something
25    #
    # Port_Type is indexed by verilog (input,output,inout,reg,wire) its
result is
    # a "," separated list of all ports of that type
30    #
    # Port_Width is indexed by the port name. Its result is "left,right"
where
    # left is the first dimension of the array, right is the last dimension.
    # eg. for wire [7:3] foo; $Port_Width{"foo"} = "7,3";
35    #
    # Port_List is indexed by the module port_name.
    # Its value is SIGNAL $port_name : (IN|OUT|INOUT) STD_LOGIC(_VECTOR?)
(left DOWNT0 right)
40    #
    # Signal_List is indexed by the register/wire name
    # Its value is $Signal_List{$port} = "SIGNAL $port : STD_LOGIC(_VECTOR?)
(left DOWNT0 right)"

    undef %Signal_List;
45    my %Signal_List;

    undef %Type_List;
    my %Type_List;

50    my %Memory_Type;

    undef %Port_List;
    undef %Port_Width;
    undef %Port_Type;

55    my %Port_List;
    my %Port_Width;
    my %Port_Type;
    if (0) #($pass == 2)
60    {
        my $Port_Width_Pointer =
$$Module_Indexed_Port_Width_Pointer($module_name);
        die "ERROR MODULE ($module_name) HAS NOT BEEN PREVIOUSLY SCANNED\n"
```

```

if ($Port_Width_Pointer eq "");
foreach $key (sort (keys(%{$Port_Width_Pointer})))
{
    #warn "module_name ($module_name) key ($key)\n";
    $Port_Width{$key} = "Taken\n";
}
}
$Module_Indexed_Port_Names_Pointer,
$Module_Indexed_Parameter_List_Pointer,

my @Module_Info = (%Port_Width,%Signal_List,%Wire_Assignments);
my $counter = 0;
my $number_of_commands = 25;
while ($module_commands =~ s/^(.*?)\;(.*?)$2/s)
{
    #Counter prints "." after $number_of_commands
    $counter++;
    print STDERR "."
        if (($counter % $number_of_commands) == 0);

    my $this_command = $1;
    my $next_commands = $2;
    my $rest_of_module = "$1\n;$2";
    #warn "rom is $rest_of_module \n";
    #warn "tc, $this_command\n";
    #Search for parameters and determine their type,
    #Types are "NATURAL" if they only contain numbers, else Type is
"STRING"
    if ($this_command =~ /^(.*?)\bparameter\s+(.*?)\s*$/s)
    {
        my $parameter_equation;
        $parameter_equation = $2;
        my ($param_lhs,@param_rhs) = split
(/\\s*\\=\\s*/,$parameter_equation);
        my ($param_rhs) = join ("\\=",@param_rhs);
        my ($parameter_type) = "STRING";
        $parameter_type = "NATURAL"
            if ($param_rhs =~ s/^(\\s*(\\d+)\\s*/$1/s);
        #warn "$param_lhs,$param_rhs,pt is $parameter_type\n";
        #was $Parameter_String .= "      $param_lhs : $parameter_type :=
$param_rhs\n";
        $Parameter_List{$param_lhs} = "$parameter_type := $param_rhs";
        #warn "found paramter ".$Parameter_List{$param_lhs}."\n";
    }

    #####
    # Get Signal Definitions
        if ($this_command =~
/(.*?)\\b(input|output|inout|reg|wire|integer)\\b(.*?)s)
    {
        my $direction = $2;
        my $port_list = $3;
        my $left_index = "";
        my $right_index = "";
        my $width;

        #####
        # (Signal Widths Width > 1) => std_logic_vector
        if ($port_list =~ s/^(\\s*\\[[^\\:]+\\]\\:([\\^\\]]+))\\s*(.*?)$3/)
        {
            $left_index = eval($1);
            $right_index = eval($2);

```

09380106.061201

```

$stype =
"STD_LOGIC_VECTOR".&Vector_Order($left_index,$right_index);
$width = "$left_index,$right_index";
}
5 else #Width is one.
{
    if ($direction =~ /^inout$/)
    {
        #all inouts of width one should be considered
10 #std_logic_vector.
        $stype = "STD_LOGIC_VECTOR (0 DOWNT0 0)";
        $width = "0,0";
    }
    else
15 {
        if ($direction =~ /^integer$/i)
        {
            #Convert integers to 32 bit STD_LOGIC_VECTORS
            $width = "31,0";
20 $stype = "STD_LOGIC_VECTOR(31 DOWNT0 0)";
        }
        else
        {
            #####
25 # everything else is STD_LOGIC.
            # (wires and registers which will
            # be converted back to STD_LOGIC_VECTOR in the if ($dir)
condition below.)
            # Perhaps this could be cleaned up later
30 $stype = "STD_LOGIC";
            $width = "0,0";
        }
    }
}
35 #####
# Wires can be declared and assigned to in the same statement
# e.g. wire a = b + c;
# So we strip away everything after the first "=" assignment
40 # And put the results in $Wire_Assignments
my $rhs;
if ($direction eq "wire")
{
    my @rhs = ();
45 ($port_list,@rhs) = split (/s*\={1}\s*/s,$port_list);
    $rhs = join ("=",@rhs); #Put $rhs back together again
}

#####
50 # Handle memories e.g.
# reg [7:0] mem_array [512 - 1 : 0];
if ($port_list =~ s/^(.*?)\[([.*?)]\:([.*?)]\s*$/\1/s)
{
    my $up_index = eval($2);
55 my $down_index = eval($3);
    $width .= ", $up_index, $down_index";
    my $array_order = &Vector_Order($up_index, $down_index);
    my $mem_type = "memory_type_$array_order";
    while ($mem_type =~ s/\s\/\s/){;}
60 my $memory_type =
&Get_Exclusive_VHDL_Name($mem_type, \%Port_Width);
    $Type_List{$memory_type} = "TYPE $memory_type IS ARRAY
$array_order of $type;";

```

0900106"061201

```

    $type = $memory_type;
}

#####
# Get Port Names and transform names to Port/Signal_List
$port_list =~ s/^\s*(.*?)\s*$/1/s; #Strip spaces at either end
foreach $port (split(/\s*\s/, $port_list))
{
    die "ILLEGAL PORT NAME $port, $1\n" if ($port =~ /(\W)/);
    $Port_Type{$direction} .= "$port,";
    $Port_Width{$port} = $width;

    #####
    # If direction is a port, put it in port_list

    if ( ($direction eq "input") ||
        ($direction eq "output") ||
        ($direction eq "inout")
        )
    {
        my %dir_xform;
        $dir_xform{"input"} = "IN";
        $dir_xform{"output"} = "OUT";
        $dir_xform{"inout"} = "INOUT";
        $dir_xform{"reg"} = "";
        $dir_xform{"wire"} = "";

        my $dir = $dir_xform{$direction};
        $Port_List{$port} = "SIGNAL $port : $dir $type";
    }

    #####
    # If direction is an internal signal or (an output which must
    # signal associated with it) or (an input port of STD_LOGIC)
    # declare it in %Signal_List

    if ( ($direction eq "wire") ||
        ($direction eq "reg") ||
        ($direction eq "integer") ||
        ($direction eq "output") ||
        ( ($direction eq "input") && ($type eq "STD_LOGIC") )
        )
    {
        # Convert all STD_LOGIC to STD_LOGIC_VECTOR(0 DOWNT0 0)
        my $tmp_type = $type;
        $tmp_type =~ s/^STD_LOGIC$/STD_LOGIC_VECTOR(0 DOWNT0 0)/i;

        #####
        # Do not put a wire/register in signal list if it has already
        # been declared in port list.

        $Signal_List{$port} = "SIGNAL $port : $tmp_type;"
            unless $Port_List{$port};

        #wire foo = some value; // Put (foo = some value) in wire
        assignments
        if ( ($direction eq "wire") && ($rhs ne "") )
        {
            my $wire_assignment = &V2VHD_Equation_Wrapper ("$port =
            $rhs", @Module_Info);
            $Wire_Assignments .= " $wire_assignment;\n";
        }
    }
}

```

09880106 "061201

```

    }
    if ( $direction eq "input" || $direction eq "output")
    {
5      #####
      # VHDL will not allow you to make equations
      # with outputs in the equation rhs. So we
      # make a wire called tmp_$port for each output named
10     $port.
      # later we assign the output $port <= tmp_$port and
      change all
      # assignments in the module to use
      # tmp_$port instead of port.

15     my $tmp_port =
      &Get_Exclusive_VHDL_Name("tmp_$port",\%Port_Width);
      #keep $port as the key so we can wire up tmp_$port and
      $port later
      $Signal_List{$port} = "SIGNAL $tmp_port : $tmp_type;";
20     #warn "added ($Signal_List{$port}) in addition to
      portlist ($Port_List{$port})\n";
    }
  }
25  }
  #next if ($pass == 1);
  #cannot do next until we've determined that the entity is a black box

  if ($this_command =~ /^(.*)\bassign\b(.*)\b(.*?)\b(.*?)$/s)
30  {
    my $wire_assignment = "    ".$V2VHD_Equation_Wrapper (" $2 =
    $3",@Module_Info)."\n";
    $Wire_Assignments .= $wire_assignment;
  }

35  #get module instantiation
  if ($this_command =~ /(.*?)\b(\w+)\b\s+(\w+)\b\s*(((.*))\b\s*$/s)
  {
    my $module_being_instantiated = $2;
    my $instance_name = $3;
    my $Connections = $4;

40    #####
    # hack a register into syn_dpram
    if ($module_being_instantiated =~ /^syn_dpram\b(\d+)\b(\d+)\b_\ruwr$/i)
    {
      #warn "warning, found syn_dpram\n";
      my $WrAddress = $Connections;
      my $WrClock = $Connections;

50      die "WrAddress not found for LPM_ROM $instance_name"
        unless ($WrAddress =~
s/^(.*?)\b\s*WrAddress\b\s*\b(.*?)\b\s*$/s);
      die "WrClock not found for LPM_ROM $instance_name"
        unless ($WrClock =~
55 s/^(.*?)\b\s*WrClock\b\s*\b(.*?)\b\s*$/s);
      my $Delayed_Address = &Add_Intermediate_Signal
        ("d1_$WrAddress",
60      &Width_Of($WrAddress,\%Port_Width),
        @Module_Info);

      $Connections =~ s/\b$WrAddress\b/$Delayed_Address/;

```



```

$Process_Statements .= "--GENMEM WARNING!!! SUPER HACK MADE FOR
VERSION 1.0 SIMULATION WARNING!!! WARNING!!!\n";
$Process_Statements .= "--GENMEM of
5 $module_being_instantiated\.vhd DOES NOT CORRECTLY LATCH WrAddress\n";
$Process_Statements .= "--SO WE HACK IN A LATCH HERE AND WIRE
$module_being_instantiated\.WrAddress to $Delayed_Address\n";
$Process_Statements .= "PROCESS\n BEGIN\n WAIT UNTIL
$WrClock = \"1\";\n $Delayed_Address <= $WrAddress;\n";
10 $Process_Statements .= "END PROCESS;\n";
    die "using memory\n";
}

    if ($module_being_instantiated =~ /^LCELL$/i)
15 {
        $Connections =~ s/\s+//gs;
        my ($lhs,$rhs) = split (/\/s,$Connections);
        $Wire_Assignments .= "--LCELL $lhs = $rhs\n";
        $Wire_Assignments .= "    .&V2VHD_Equation_Wrapper ($lhs =
20 $rhs",@Module_Info)."\;\n";
    }
    else
    {
        #warn "VERILOG TO VHDL CONVERSION: ".&date_time." :
25 INSTANTIATING $instance_name IN MODULE $module_name\n";
        #If module_being_instantiated is a black box, declare it as
a component,
        #else instantiate it normally

30        #fpga express likes everything to be a component. All
other files accept ENTITY work and only use component as a black box.
        #forcing if (1) makes fpga express
        #happy. component_list_pointer makes everyone else happy
        #if (1)
35        if ($$Component_List_Pointer{$module_being_instantiated})
        {
            $Declared_Components{$module_being_instantiated}++;
            $instantiation_string .= "$instance_name
$module_being_instantiated\n";
40        }
        else
        {
            if ($pass == 2)
            {
25        $Module_Instantiation_List{$module_name} .=
"$module_being_instantiated,"
                unless ($Module_Instantiation_List{$module_name}
== /\b$module_being_instantiated\./);
50        $instantiation_string .= "$instance_name : ENTITY
work.$module_being_instantiated\ (behavior\)\n";
            }
        }

55        if ($DefParam_List{$instance_name})
        {
            $instantiation_string .= "    GENERIC MAP
\\(\n".$DefParam_List{$instance_name})."    \)\n";
            #warn "PARAMETERS in
60 $instance_name\n".$DefParam_List{$instance_name})."\n";
        }
    }

```

#####

```

# I initially thought it would be really easy to just hook
up the ports. 95% of the time it is total cake.
# But 5% of the time is a royal pain.
#
# Here are the major differences between verilog and VHDL
port instantiation.
#
# 1) Verilog does not care that you leave a port
unconnected. VHDL does.
# 2) Verilog does not care that the widths of the ports and
the widths of the connected signal
# do not match. VHDL does.
# 3) Verilog considers a bit vector of width 0 and a single
bit to be interchangeable.
# VHDL considers STD_LOGIC different than
STD_LOGIC_VECTOR(0 DOWNT0 0).
#
# Problem 1 is easy to solve: Keep a list of module ports
and instantiated connections
# Each Port that does not have an associated connection
gets wired to "open".
#
# Problem 2 is solved in the following manner:
# If a port is an input, e.g. ".input_signal ({~a ,
{4(b)}})",
# use the results from our
V2VHD_Equation_Wrapper(input_signal = ({~a , {4(b)}}...)). V2VHD_Equation
# is sophisticated enough to deal with all verilog
operators and can also reconcile widths.
#
# For each output connection, e.g. ".output_signal ({~a ,
{4(b)}})"
# We make a tmp_output_signal which has the same width as
the output port. Then we use V2VHD_Equation
# again, &V2VHD_Equation_Wrapper("({~a , {4(b)}}) =
tmp_output_signal"...)).
#
# Problem 3 is also solved by having a tmp signal act as the
go-between.
#
# This makes for very ugly vhd code. When I have time,
I'll fix it so that the
# only time we generate a tmp signal is for the nasty 5% of
the time.

my $Module_Port_Names_Pointer =
$$Module_Indexed_Port_Names_Pointer{$module_being_instantiated};
die "ERROR IN INSTANTIATING ($module_being_instantiated).
UNKNOWN MODULE\n"
if ( ($Module_Port_Names_Pointer eq "") && ($pass ==
2));
my $Module_Port_Width_Pointer =
$$Module_Indexed_Port_Width_Pointer{$module_being_instantiated};
my $Module_Parameter_Pointer =
$$Module_Indexed_Parameter_List_Pointer{$module_being_instantiated};

my %Connection_List;

foreach $connection (split (/\\s*\\,\\s*/s,$Connections))
{
#last if ($pass == 1);

```

09330106 "061201

```

        if ($connection =~ /\.\s*(\w+)(.*)/S)
        {
            my $Module_Port_Name;
            $Module_Port_Name = $1;
            #my $Connection_rhs = $2;
            my ($tmp,$Connection_rhs,$tmp2) =
5      &Count_Parentheses($2);

            die "ERROR INSTANTIATION OF $instance_name IN MODULE
10  $module_name NOT UNDERSTOOD ($connection)$1\n"
                if ($Connection_rhs eq "");
            $Connection_rhs =~ s/^\s*(.*?)\s*$/$1/;
            $Connection_List{$Module_Port_Name} = $Connection_rhs;
            #warn "INSTANTIATION ($module_being_instantiated)
15  adding $Connection_rhs, to
            $Module_Port_Name(".$Connection_List{$Module_Port_Name}."\n";
            die "ERROR INSTANTIATION OF MODULE
            $module_being_instantiated NAMED $instance_name\n"
            ." IN MODULE $module_name REFERS TO AN UNKNOWN
20  PORT $Module_Port_Name\n"
                unless
                (($Module_Port_Width_Pointer{$Module_Port_Name} || ($pass == 1));
                }
            else
            {
25          die "ERROR INSTANTIATION OF $instance_name IN MODULE
            $module_name NOT UNDERSTOOD ($connection)$1\n";
            # $module_port_list .= " $connection,\n";
            }
30      }

        my $module_port_list;
        my $pw_array = join ("\n ",keys(%Port_Width));

35      #loop over ports of module being instantiated
        foreach $port (sort(keys(%$Module_Port_Names_Pointer)))
        {
            my $what_port_connects_to = $Connection_List{$port};
            if ($what_port_connects_to eq "")
40          {
                $module_port_list .= " $port => open,\n";
                next;
            }
            my $port_width = &Width_Of($port,
45          $Module_Port_Width_Pointer);

            my ($tmp_name,
                $direction,
                $port_type) =
50      &Get_Port_Name_Direction_And_Type($Module_Port_Names_Pointer{$port});

            my %E_List;
            my $port_name =
55      &V2VHD_Equation($what_port_connects_to,@Module_Info,\%E_List);
            $port_name =~ s/^\s*(.*?)\s*$/$1/s;

            my $connection_width =
            &Width_Of($port_name,\%Port_Width);

60      if ( $connection_width eq "" && ($direction =~ /^OUT$/))
            {

```

```

warn "WARNING:  INSTANTIATION OF $instance_name IN
MODULE $module_name HAS A PORT NAMED\n";
warn "          ($what_port_connects_to) THAT HAS NOT
BEEN DEFINED.  ASSUMING A SIGNAL OF WIDTH\n";
5  warn "          ($port_width) EQUIVALENT TO WIDTH OF
MODULE ($module_being_instantiated) PORT\n";
warn "          ($port_name)\n";
$connection_width = $port_width;

10  &Add_Intermediate_Signal($what_port_connects_to,$connection_width,@Module_Info)
;
    }

    if ($E_List($what_port_connects_to))
15  {
        $port_width =
&Width_Of($what_port_connects_to,\%Port_Width);
    }

20  #If the port width does not match or if
$what_port_connects_to contains some funky operators
    #do the safe thing and make a tmp_wire.

    if (($connection_width != $port_width) ||
25  ($what_port_connects_to =~ /\W/))
    {
        if ($direction =~ /^INOUT$/i)
        {
            foreach $a (keys (%Port_Width))
30  {
                print "cw $a, $Port_Width{$a}\n";
            }

            foreach $foo (keys (%$Module_Port_Width_Pointer))
35  {
                print "here is module $foo,
$$Module_Port_Width_Pointer{$foo}\n";
            }
            die ("ERROR MODULE $module_name, INSTANTIATION
40  $instance_name INOUT PORT, \n"
                ."$what_port_connects_to (width $connection_width)
                MUST HAVE THE SAME WIDTH AS $port (width $port_width)\n");
        }

45  if ($direction =~ /^IN$/i)
    {
        $what_port_connects_to = "To_$instance_name\_$_port";
        $what_port_connects_to =
&Get_Exclusive_VHDL_Name($what_port_connects_to,\%Port_Width);
50  #warn "instance_name is $instance_name, port is $port,
mpwn is ".$$Module_Port_Width_Pointer{$port}."\n";
        $Port_Width{$what_port_connects_to} =
        $$Module_Port_Width_Pointer{$port};
        $Signal_List{$what_port_connects_to} = &Declare_Signal
55  ($what_port_connects_to,\%Port_Width);

        $Wire_Assignments .= "      "&V2VHD_Equation_Wrapper
        ("$what_port_connects_to = ".$Connection_List{$port},

60  @Module_Info)."\;\n"

        if ($pass == 2);
    }
    if ($direction =~ /^OUT$/i)

```

```

    {
        $what_port_connects_to = "From_$instance_name\_$_port";
        $what_port_connects_to =
5      &Get_Exclusive_VHDL_Name($what_port_connects_to,\%Port_Width);
        $Port_Width($what_port_connects_to) =
        $$Module_Port_Width_Pointer($port);
        $Signal_List($what_port_connects_to) = &Declare_Signal
        ($what_port_connects_to,\%Port_Width);

10      $Wire_Assignments .= "    ".$V2VHD_Equation_Wrapper
        ($Connection_List($port)." = $what_port_connects_to" ,

        @Module_Info)."\;\n"
        if ($pass == 2);

15      }
    }
    $what_port_connects_to .= "(0)" if ($port_type =~
/^STD_LOGIC$/i);
    $module_port_list .= "    $port =>
20    $what_port_connects_to,\n";
    }
    #Lose the last comma
    $module_port_list =~ s/\,\s*$//s;

25    $instantiation_string .= "    PORT MAP \(\n";
    $instantiation_string .= "$module_port_list    \)\;\n\n";
    }
}

30 #####
# In verilog, you often declare a bunch of wire and assign statements
# right before you do an always or initial block. It's nice to do it
this way
# so that wires that determine the outcome of the always statement
35 # are near the actual always statement.
#
# We do the same thing in our initial and always blocks.
# We put all previously assigned wires
# before the PROCESS statement. And clear out $Wire_Assignments so
40 # that the assignments only show up in one place.
#
# find initial always statements.
#
# assume that a statement always @(posedge a or posedge b)
45 # always has "a" as the clock and "b" as the asynchronous event.
# we can get fancier later.

if ($rest_of_module =~ /\s*\b(always|initial)\b\s*(.*)$/s)
{
50     #update heartbeat
    print STDERR ".";
    $counter = $number_of_commands;

    #warn "in always statement\n";
    my $always_or_initial = $1;
    my $always_statement = $2;
    #warn "found always statement, $always_statement\n";
    my $clk;
    my $clk_level = "0";
    my $edge;
    my $asynchronous_event;
    my $asynchronous_level = "0";
    my $asynchronous_edge;
60

```

my \$wait_statement;

my \$tmp;
my \$always_condition;
my \$always_innards = \$2;
my \$tmp_always_condition;

\$Process_Statements .= \$Wire_Assignments;
\$Wire_Assignments = "";

#Search for always @(foo)
if (\$always_statement =~ /^@(.*?)\$/s)
{

 #warn "found \@ remaining \$1\n";
 (\$tmp,\$always_condition,\$always_innards) =

&Count_Parentheses(\$1);

 #convert always condition
 \$tmp_always_condition = \$always_condition;

 #get clock and clocks edge
 if (\$always_condition =~ s/(pos|neg)edge\s+(\S+)(.*)/\$3/s)
 {

 \$clk = \$2;
 \$edge = \$1;
 \$clk_level = "1" if (\$edge eq "pos");
 \$wait_statement = "UNTIL \$clk = \'".(\$edge eq "pos")."\'";
 \$wait_statement = "UNTIL \$clk = \'" . (\$edge eq "pos")."\'";

 #warn "always condition is (\$always_condition)\n";

 }
 else
 {

 if (\$always_or_initial =~ /always/i)

 {
 #No clock statement, but possibly a conditional always
 #e.g. always @(a or b or c)
 while (\$always_condition =~ s/\s*\bor\b/\./i){}
 \$wait_statement = "ON \$always_condition";
 #warn "wait_statement is \$wait_statement\n";

 }
 }

}

my \$rest_of_module_innards;
(\$always_innards,\$rest_of_module_innards) =

&Handle_Next_If_Else_Line(\$always_innards,

@Module_Info);

my %Variable_Conversion_List;
while (\$always_innards =~ s/Please Convert (\w+) To A Variable//s)
{

 #warn "found conversion of (\$1)\n";
 \$Variable_Conversion_List{\$1}++;

foreach \$VCL (keys(%Variable_Conversion_List))

{
 #warn "VCL is (\$VCL), width of is

".&Width_Of(\$VCL,\%Port_Width)." \n";

 #my \$tmp_name = &Add_Intermediate_Signal("VARIABLE_\$VCL",
 #&Width_Of(\$VCL,\%Port_Width),

```

#@Module_Info);

#add intermediate signal, but don't use function since
# its possible the signal is a memory
5   my $tmp_name =
      &Get_Exclusive_VHDL_Name("VARIABLE_$VCL", \%Port_Width);

      my $tmp_signal_list = $Signal_List{$VCL};
      $tmp_signal_list =~ s/\b$VCL\b/$tmp_name/g;
10   $Signal_List{$tmp_name} = $tmp_signal_list;
      $Port_Width{$tmp_name} = $Port_Width{$VCL};
      #warn "vcl $VCL, tmp_name $tmp_name, width
      # $Port_Width{$tmp_name}, signal $Signal_List{$tmp_name}\n";

15   &Convert_Signals_To_Shared_Variable($tmp_name, \%Signal_List);

      while ($always_innards =~ s/\b$VCL\b/$tmp_name/si){;}
      $always_innards = "      $tmp_name := $VCL;\n$always_innards\n
$VCL <= $tmp_name;";
20   }

      #####
      # &Handle_Next_If_Else_Line has converted everything inside the
always statement to always_innards and the rest
      # of the module is in $rest_of_module_innards; So set our top
25   level module_commands = $rest_of_module_innards
      # and continue parsing from there.
      $module_commands = $rest_of_module_innards;

30   #get asynchronous signal and edge if it exists
      if ($always_condition =~ /\s+or\s+(pos|neg)edge\s+(\S+)/s)
      {
          $asynchronous_edge = $1;
          $asynchronous_event = $2;
35   $asynchronous_level = "1" if ($asynchronous_event eq "pos");

          #my $first_if_then_statement = "IF $asynchronous_event =
          \'$asynchronous_level\' THEN";
          my $first_if_then_statement = "IF $asynchronous_event =
40   \'$asynchronous_level\' THEN";

          my $death_string = "ERROR ALWAYS CONDITION
$tmp_always_condition HAS ASYNCHRONOUS SIGNAL ($asynchronous_event)\n"
          ."BUT DOES NOT HAVE THE SIGNAL IN THE FIRST IF STATEMENT.\n"
45   ."INSIDE ALWAYS BLOCK IS ($always_innards).\n";

          #Process first if condition
          #make sure asynchronous edge was involved in first if
computation.
50   die ("$death_string") unless ($always_innards =~
s/^(\\s*)IF(.*?)THEN(.*?)$/\\$1$first_if_then_statement$3/s);
          my $fic = $2;
          die ("$death_string") unless ($fic =~ /\$asynchronous_event/s);
55   die ("$death_string") unless
($always_innards
s/($first_if_then_statement.*?)ELSE(.*?)$/\\$1ELSIF $clk\'EVENT AND $clk =
\'$clk_level\' THEN$2/s);

60   $Process_Statements .= "PROCESS ($clk,$asynchronous_event)\n
BEGIN";
      $Process_Statements .= "      ";
      }

```

```

else
{
    $Process_Statements .= "PROCESS\n BEGIN\n";
    $Process_Statements .= "    WAIT $wait_statement;\n"
5      if ($wait_statement);
    #$Process_Statements .= "$always_innards \n END PROCESS;\n";
}

    $Process_Statements .= $always_innards;
10    $Process_Statements .= "\n    WAIT;"
    if ($always_or_initial =~ /initial/);
    $Process_Statements .= "\nEND PROCESS;\n\n";
} #Done with always @innards.
}
15 #If we've printed status ".", print a new line
print STDERR "\n"
    if ($counter >= $number_of_commands);

#####
20 # Put it all together.
#
# Check to see if anything needs to be put inside the architecture block.
# If nothing does, assume its a black box and instantiate a "component"
with noopt
25 # attributes and the exact same Parameters and Ports as the entity hooked
up to the
# component.
# P.S. I hate black boxes.
if
30 (($Process_Statements.$Wire_Assignments.$instantiation_string.$attribute_string
) eq "")
{
    my $Component_String =
&Declare_Entity($module_name,\%Port_List,\%Parameter_List);
35 while ($Component_String =~ s/ENTITY/COMPONENT/){;}
    while ($Component_String =~ s/END\s+$module_name\s*/;/END
COMPONENT/;){;}
    my $tmp_cs = $Component_String;
    $Component_String .= " --attribute noopt: boolean;\n";
40 $Component_String .= " attribute noopt of $module_name: component is
true;\n";
    $Component_String .= " --Hard instantiation of $module_name
megafunction in VHDL with user defined parameters\n\n\n";
    $$Component_List_Pointer{$module_name} = $Component_String;
45 }
    #else #It is not a black box, declare it as a normal module
    if
    (($Process_Statements.$Wire_Assignments.$instantiation_string.$attribute_string
) ne "")
50 {
    #####
    # Unlike Verilog, VHDL requires that all entities be defined before
another
    # architecture block instantiates said entity. (Sounds like a legal
55 verdict doesn't it?)
    # So, we put our port and parameter information into
$entity_declaration, then put all
    # $entity_declaration at the top of our file.

    my $entity_declaration =
60 &Declare_Entity($module_name,\%Port_List,\%Parameter_List);

    #Everything below goes in the architecture block.

```


#I'll put a commented out entity declaration in there too so
 #it will be easier for a coder to figure out what is going on
 #Put it all in a string called \$architecture_block

```

5      my $architecture_block = "ARCHITECTURE behavior OF $module_name
IS\n";
      my (@Component_Array) = sort (keys (%Declared_Components));
      $architecture_block .= "attribute noopt: boolean;\n" if
@Component_Array;
10     foreach $key (@Component_Array)
    {
      $architecture_block .= ($$Component_List_Pointer{$key});
    }

15     foreach $type (sort(keys(%Type_List)))
    {
      $architecture_block .= "      ".$Type_List{$type}."\n";
    }

20     while ($Wire_Assignments =~ s/\`//g){;} #crush tick escape character
should already be done, but it does not hurt anything
      #warn "wa is $Wire_Assignments\n";

      #####
25     # VHDL will not allow you to make equations
      # with outputs in the equation rhs. So we
      # make a wire called tmp_$port for each output
      # and assign the output $port <= tmp_$port. All
      # other assignments in the module are changed to use
30     # tmp_$port instead of port.
      #
      # Also, VHDL considers std_logic to be different than
      std_logic_vector(0 downto 0). Verilog does not.
      # We solve this above by assuming everything is a std_logic_vector(0
35     downto 0). Here we'll take
      # a port (std_logic) and instantly convert it to tmp_port
      (std_logic_vector(0 downto 0)) and set
      # tmp_port(0) <= port; if port is input and port <= tmp_port(0) if
      port is output.
40     # Extra tricky, though is that fpga express does not like 'event on
      std_logic_vectors.
      # we'll need to convert 'event signals back to std_logic

      foreach $port_declaration (sort(keys(%Port_List)))
45     {
      my ($port_name,$port_direction,$port_type) =

      &Get_Port_Name_Direction_And_Type($Port_List{$port_declaration});

50     my $port_type_is_std_logic = 0;
      $port_type_is_std_logic = 1
      if ($port_type =~ s/^STD_LOGIC$/STD_LOGIC_VECTOR(0 DOWNT0
0)/i);

55     my $tmp_signal_list = $Signal_List{$port_declaration};
      if (
      $tmp_signal_list
      =~
      /\^s*(SIGNAL|VARIABLE|SHARED\s+VARIABLE)\s*(\w+)/is
      #($port_direction =~ /\^OUT$/i) ||
      #($port_type_is_std_logic)
60     )
      {
      my $tmp_port_name = $2;

```

09380106 "061201

```

#warn "port_name is $port_name tmp_port_name is $1,
$tmp_port_name\n";
#add new name to signal list if any Process,Wire, or
instantiation signal uses it.
my $port_name_used = 0;
#Orion, maybe could use /g option here if we knew the special
variable for how many
#times /g matched or we could next if we were here for pass 1

while ($Process_Statements ==
s/\b$port_name\b/$tmp_port_name/s){$port_name_used++;}

my $event_Process_Statements;
#Remember, clk is first in an asynchronous reset statement.
if ($port_name_used)
{
while ($Process_Statements =~
s/\b(PROCESS\s*\(\s*)
reset\)"
statement
'event and
" $6 = new clk logic level
to end process

$1$port_name$2$3$port_name$4$port_name$5\'$6\'$7/six)

{
$port_name_used = $port_name_used - 2;
die "Incorrect \'event substitution\n"
if ($port_name_used < 0);
}

while ($Process_Statements ==
s/(\bPROCESS\s+BEGIN\s+WAIT\s+UNTIL\s+)\$tmp_port_name(\s+=\s+)\\"([01])\\"/
$1$port_name$2\'$3\'/six)
{$port_name_used = $port_name_used - 1;}
}

while ($Wire_Assignments ==
s/\b$port_name\b/$tmp_port_name/s){$port_name_used++;}

while ($instantiation_string ==
s/(\\=\\>\\.\\*?\\b)$port_name\\b/$1$tmp_port_name/){$port_name_used++;}

if ($port_name_used)
{
my $tmp_wire_assignment;
$tmp_port_name .= "(0)" if ($port_type_is_std_logic);
if ($port_direction =~ /^OUT$/){$tmp_wire_assignment = "
$port_name <= $tmp_port_name;\n";}
if ($port_direction =~ /^IN$/){$tmp_wire_assignment = "
$tmp_port_name <= $port_name;\n";}
#die "DID NOT FIND ($port_name) IN ARCHITECTURE BLOCK
($code_in_architecture_block)\n"
#unless ($code_in_architecture_block ==
s/(\\.\\*\\[\\^\\;\\=\\]\\*\\b$port_name\\b\\[\\^\\;\\=\\]\\*\\[\\^\\;\\]\\*\\;\\)(\\.\\*)/$1$tmp_wire_assignment$2/s);
$Wire_Assignments .= $tmp_wire_assignment;
}
else
{

```

```

        undef ($Signal_List($port_name));
    }
}
#####
# Now because FPGA express hates std_logic_vector 'event and wait
statements, we need to generate
# a std_logic signal to be used in all other 'event and wait
statements
my %std_logic_xform;
if (0) #No, I refuse to support FPGA express
{
    while ($Process_Statements =~
        s/\b(PROCESS\s*\(\s*)
            (\w*)(\s*\,\s*\w+\s*\))
                # $1 = PROCESS \(
                    # $2 = clk, $3 = "
, reset\)")
        (.*)\b)
            # $4 = reset
statement
        \2(\s*'EVENT\s+AND\s+)
            # $5 = 'event and
        \2(\s+=\s+)\\"([01])\\"
            # $6 = " = " $7 = new clk
logic level
        (\s+.*)\bEND\s+PROCESS)
            # $8 = rest to
end process
        /$1$2$3$4($2(0))$5$2(0)$6\'$7\'$8/six)

    {;}
    #my                                $std_logic_name              =
    "($2(0))";#&Get_Exclusive_VHDL_Name(std_logic_$2,\"%Port_Width");
    #std_logic_xform{$1} = $2;

30   #$1std_logic_name$3$4std_logic_name$5std_logic_name$6\'$7\'$8/sixee)

        while                      ($Process_Statements              ==
s/(\bPROCESS\s+BEGIN\s+WAIT\s+UNTIL\s+)$tmp_port_name(\s+=\s+)\\"([01])\\"/
        $1$port_name$2\'$3\'/six)
35         {;}
    )

    foreach $signal (sort(keys(%Signal_List)))
40     {
        $architecture_block .= "      ".$Signal_List{$signal}."\n";
    }

    #if architecture is of cpu_register_ram, hack in a
    #substitute. this is only for nios 1.1 kits
    if ($module_name =~ /cpu_register_ram$/i)
    {
        my $new_signal = $Port_List{wraddress};
        $new_signal =~
50         s/\bwaddress\b\s*:\s*in\s+/last_wraddress \:/i;

        my @type_array = (sort keys %Type_List);
        die "ERROR Expecting only one memory in $module_name"
            if (scalar (@type_array) > 1);

55         my $memory_signal_name;
        foreach $sig (sort keys %Signal_List)
        {
            if ($Signal_List{$sig} =~ /\:\s*$type_array[0]\s*\;\s*$/)
60             {
                $memory_signal_name = $sig;
                last;
            }

```

```

    }
    die "ERROR could not find memory in $module_name\n"
    unless ($memory_signal_name);
    $Process_Statements = &Replace_Cpu_Register_Ram
5      ($memory_signal_name);

    $architecture_block .= "    $new_signal;\n";

    $Wire_Assignments = "";
10  }

#####
# there may some initial statements of variables that
# are written by another process.  Solution: suck up
# initial processes and ensure that values aren't being
15 # assigned to in other processes.  If they are, put the
# initial assignments in the other processes

my @initial_statements = ();
20 while ($Process_Statements =~
    s/^(.*)process
        \s+begin\s+
        (.*?)\s*
        \bwait\s*\;\s*
25        end\s+process\s*\;
        (.*?)
        /\$1/six)
    {
        my ($initial_assignments,
30          $rest) = ($2,$3);

        foreach $assignment (split (/\\s*\;\s*/s,
            $initial_assignments))
        {
35          my ($lhs,
              $assign_operator,
              $rhs) =
              $assignment =~ /\^\\s*(.*?)\\s*([\\<\\:]=)\\s*(.*)/s;

          next unless ($lhs);

          #do not support concatenations for now.
          #also assume that indexes are not split across
          #process statements.

          die "too many lhs arguments in $lhs of assignment",
45          " $assignment\n"
              if ($lhs =~ /\./);

          # now we have a single lhs argument, search around
          # for the appropriate always statement.  The
          # statement could be in the previously modified
          # $Process_Statements or in $rest.  So search in
          # each of them.

          if ($lhs =~ /\(/)
          {
50              my @tmp;
              ($lhs,@tmp) = &Count_Parentheses($lhs);
          }
          foreach $string (\$Process_Statements,$rest)
          {
              if ($$string =~
55              
```

```

s/(.*\bprocess\s+
begin\s+.*?\b)

(\s*loop\s+.*?
(\bthen\s+|\belse\s+|\;;\s*)
($lhs\b[^;]*([\<:\;]=).*?;)
)
/
"$1".
"$assignment\;".
"\n$2"
/sixe
)
{
die "mismatched assignment operators\n",
"initial statement was $assignment\n",
"always assign was $4\n"
if ($assign_operator ne $5);
last;
}
else #check for no loop:
{
if ($$string =~
s/(.*\bprocess\s+
begin\s+)
(.*)
(\bthen\s+|\belse\s+|\;;\s*)
($lhs\b[^;]*([\<:\;]=).*?;)
.*?
)\s*(\bend\s+process\s*\;;)
/
"$1".
"\n$assignment\;".
"\nLOOP\n".
"$2".
"\nEND LOOP\;\n".
"$6"
/siex
)
{
die "mismatched assignment operators\n",
"initial statement was $assignment\n",
"always loop was $4\n"
if ($assign_operator ne $5);
last;
}
}
if ($string == \$rest)
{
push (@initial_statements, "$assignment;");
}
}
$Process_Statements .= $rest;
}
if (@initial_statements)
{
$Process_Statements .= "PROCESS\nBEGIN\n";
foreach $is (@initial_statements)
{
$Process_Statements .= "$is\n";
}
$Process_Statements .= "WAIT;\nEND PROCESS;\n";

```

09880106 061201
T02T90" 90T08860

```

    }

    $architecture_block .= "BEGIN\n\n";
    $architecture_block
5  $Process_Statements.$Wire_Assignments.$instantiation_string.$attribute_string;
    $architecture_block .= "END behavior;\n\n\n";

    #last thing convert [a:b] to (a downto b)
    $architecture_block = &V2VHD_Index($architecture_block);
10

    $Entity_And_Architecture($module_name) =
&Get_Library_Declaration($architecture_block).$entity_declaration.$architecture
_block;
15

    # $all_architecture_blocks .= $architecture_block;
    # warn "defined $module_name\n";
}
#####
20 # Now we are finished with the contents of the module.
    # Save away valuable information for later regardless of if its a black
    box or not

    $$Module_Indexed_Port_Width_Pointer($module_name) = \%Port_Width;
    $$Module_Indexed_Port_Names_Pointer($module_name) = \%Port_List;
    $$Module_Indexed_Parameter_List($module_name) = \%Parameter_List;
25

}
#####
30 # Now all is done, except that we need to order the modules
    # VHDL is particular about the order in which things are declared.
    # Everything must be declared before it is instantiated. Fortunately, I
    # have a list of what is instantiated called %Module_Instantiation_List;

35 my $return_string;

    foreach $entity (&Order_Entities(\%Module_Instantiation_List,
                                     @Module_Array)
    {
40         $return_string .= $Entity_And_Architecture($entity);
    }

    return ($return_string);
45 }

#####
#
# Order_Entities is a recursive function that takes a hash and an array of
50 keys.
# It returns an array of entities in order of dependance. The first entity
    returned
# will not instantiate any other entities. The second entity returned will
    only instantiate
55 # the first entity if it instantiates any entities. Likewise for the
    third...last
# entites. The last module returned will be the top level in the heirarchy.
#
# The value of the hash is a comma separated string of entities that are
60 instantiated
# within the hash.
# i.e.

```

```

# $$pInstantiation_Hash{a} = "first module instantiated in a, second module
instantiated in a,
# last module instantiated in a"
#
5 # @keys is an array of all entity names that should be ordered.
#####
#

sub Order_Entities
10 {
    my ($pInstantiation_Hash,@keys) = @_;
    my @return_array;

    my $already_declared = "module already declared";
15 #There is no way that we can confuse "module already declared" with a
verilog module name
#module is a key word and no spaces are allowed in module names.

    foreach $key (@keys)
20 {
        if ($$pInstantiation_Hash{$key} eq $already_declared)
        {
            #warn "$key already declared\n";
            next;
25 }
        if ($$pInstantiation_Hash{$key} ne "")
        {
            my $value = $$pInstantiation_Hash{$key};
            $value =~ s/\\,\\s*$/;
30 push (@return_array,
                &Order_Entities($pInstantiation_Hash,
                    split(/\\s*\\,\\s*/s,$value)
                )
            )
35 }
        #now everything on which $key is dependant has been declared
        #so its safe to declare it.
        push (@return_array, $key);
        $$pInstantiation_Hash{$key} = $already_declared;
40 }
    return (@return_array);
}

sub V2VHD_Index
45 {
    my ($string) = @_;
    #If a variable has two vector indecies, the second one takes precedence.
    while ($string =~ s/(\\[[^\\]]*\\)\\s*)(\\[.*?\\])/2/s){;}
    while ($string =~ s|(.*)(\\[.*?\\])(.*)|1/s)
50 {
        my $rest = $3;
        #warn "found $2\n";
        $GLOBAL_DEBUG = 1;
        $string .= &Vector_Order(split (/\\s*\\:\\s*/s,$2));
55 $GLOBAL_DEBUG = 0;
        $string .= $rest;
    }
    return ($string);
60 }

#####
#
# Declare Entity

```

```

#
# Takes a module_name, \%Port_List, \%Parameter_List and
# returns a string that contains a vhdl ready entity declaration
#
5  sub Declare_Entity
    {
        my ($module_name,$pPort_List,$pParameter_List) = @_;

        my $entity_declaration .= "ENTITY $module_name IS\n";

10     my @Parameter_Array = sort(keys(%$pParameter_List));
        if (@Parameter_Array)
        {
            $entity_declaration .= "  GENERIC \{\n";
15     foreach $parameter (@Parameter_Array)
            {
                my $Parameter_String = "                $parameter
: ".$$pParameter_List{$parameter}."; \n";
                $entity_declaration .= $Parameter_String;
20     }
            $entity_declaration =~ s/\;\n$/\n/s; #Get rid of last semi-colon.
            $entity_declaration .= "  \}; \n";
        }

25     my @Port_Array = sort(keys(%$pPort_List));
        if (@Port_Array)
        {
            $entity_declaration .= "  PORT \{\n";
30     foreach $port (@Port_Array)
            {
                $tmp = $Port_List{$port}."; \n";
                # $entity_declaration .= $tmp; # $Port_List{$port}."; \n";
                $entity_declaration .= "    ".$$pPort_List{$port}."; \n";
35     }
            $entity_declaration =~ s/(.*)\;/$/1/s;
            $entity_declaration .= "  \}; \n";
        }
        $entity_declaration .= "END $module_name;\n\n";
40     return($entity_declaration);
    }

#####
45  #
# Get_Library_Declaration
#
# Very simple for now, may get more complicated later

50  sub Get_Library_Declaration
    {
        my ($architecture_block) = @_;

        my $library_declaration =
55     "LIBRARY ieee;\n"
        . "use ieee.std_logic_1164.all;\n"
        . "use ieee.std_logic_arith.all;\n"
        . "use ieee.std_logic_unsigned.all;\n";

60     $library_declaration .= "\nlibrary std;\nuse std.textio.all;\n"
        if ($architecture_block =~ /\bwrite\(/);

        return ($library_declaration);
    }

```



```

}
sub V2VHD_Files
(
    my ($Verilog_String,$Destination_Directory,@files) = @_;
    my (@Files_To_Synthesize);
    my (@Do_Not_Synthesize_These);

    #warn "vs = $Verilog_String, dd is $Destination_Directory, files are
@files\n";
    my $COPYRIGHT_NOTICE=
"--Copyright (C) 1991-2000 Altera Corporation
--Any megafunction design, and related net list (encrypted or decrypted),
--support information, device programming or simulation file, and any other
--associated documentation or information provided by Altera or a partner
--under Altera's Megafunction Partnership Program may be used only to
--program PLD devices (but not masked PLD devices) from Altera. Any other
--use of such megafunction design, net list, support information, device
--programming or simulation file, or any other related documentation or
--information is prohibited for any other purpose, including, but not
--limited to modification, reverse engineering, de-compiling, or use with
--any other silicon devices, unless such use is explicitly licensed under
--a separate agreement with Altera or a megafunction partner. Title to
--the intellectual property, including patents, copyrights, trademarks,
--trade secrets, or maskworks, embodied in any such megafunction design,
--net list, support information, device programming or simulation file, or
--any other related documentation or information provided by Altera or a
--megafunction partner, remains with Altera, the megafunction partner, or
--their respective licensors. No other licenses, including any licenses
--needed under any third party's intellectual property, are provided herein.
";

    my %Module_Indexed_Port_Names;
    my %Module_Indexed_Port_Width;
    my %Module_Indexed_Parameter_Values;
    my %Component_List;
    my %Additional_Files;
    #warn "mipn pointer is ".$Module_Indexed_Port_Names."\n";
    my $Top_Wrapper_Name = "Top_Level_$Top_Level_Module_Name";

    $Destination_Directory =~ tr|\\|\/|;
    $Destination_Directory =~ s/^\s*(.*?)\s*$/$1/;
    $Destination_Directory =~ s/^(.*)\$/ $1/;

    foreach $pass (1,2)
    {
        warn "VERILOG TO VHDL CONVERSION: ".$date_time." PASS $pass\n";
        foreach $file (@files)
        {
            $file =~ tr|\\|\/|;
            my $vhdl_file = $file;
            $vhdl_file =~ s/^\s*(.*?)\s*$/$1/s; # Crush path. so that files in
            #warn "vhdl file is $vhdl_file\n";
            # different directories end up in the same place.
            $vhdl_file =~ s/^(.*?)\s*(.*)\s*$/$1/; #Crush all after first "."
            my $extension = $2;
            my $vhdl_innards;
            if ($extension =~ /^vhd/i)
            {
                warn "VERILOG TO VHDL CONVERSION: ".$date_time."
                " leaving vhdl file ($file) alone.\n";
            }
            else

```

```

    {
        $vhdl_file = "$Destination_Directory\"$vhdl_file\".vhd";
        warn "VERILOG TO VHDL CONVERSION: ".&date_time." SCANNING $file
\n";# TO $vhdl_file.\n";
5
        $vhdl_innards = &V2VHD($Verilog_String,
                                $file,
                                $pass,
                                \%Module_Indexed_Port_Width,
10
                                \%Module_Indexed_Port_Names,
                                \%Module_Indexed_Parameter_Values,
                                \%Component_List);
    }

15
    if ($pass == 2)
    {
        # If there is nothing in vhd_innards copy the verilog file
        directly. It will get used later in quartus.
        # otherwise, make a vhd file. No need to copy the verilog file if
20
        the output directory
        # is ".".
        if ($vhdl_innards ==~ /\s*$/s)
        {
            if (0) #($Destination_Directory ne ".")
            {
25
                open (SRCFILE, "< $file") || die "FILE ERROR! CAN NOT OPEN
$file $!\n";
                my $Dest_File = "$Destination_Directory\"$file";
                open (DESTFILE, "> $Dest_File") || die "FILE ERROR! CAN NOT
30
                OPEN $Dest_File $!\n";
                while (<SRCFILE>)
                {
                    print DESTFILE $_;
                }
                close (SRCFILE);
                close (DESTFILE);
                print "COPIED $file TO $Destination_Directory\n";
            }
        }
        else
40
        {
            open (DESTFILE, "> $vhdl_file") || die "FILE ERROR! CAN NOT
            OPEN $vhdl_file $!\n";
            print DESTFILE "$COPYRIGHT_NOTICE";
            print DESTFILE "$vhdl_innards";
            close (DESTFILE);
            warn "VERILOG TO VHDL CONVERSION: ".&date_time." CONVERTED
50
            $file TO \n".("x60")."$vhdl_file\n";
            push (@Files_To_Synthesize,$vhdl_file);
        }
    }
}
return (@Files_To_Synthesize);
55
}

sub Convert_Simulation_Files
{
    my $file = shift or die "no file specified for simulation\n";
    my $test_bench = &HDL_Remove_Comments(&HDL_Read_File($file));
60

    my @all_sim_files;

```

```

my $verilog_test_equipment;

while ($test_bench =~ s/^(.*?)\`include\s*"(.*)"/$1$3/sg)
{
5   my $include_file = $2;

    #test_equipment.v has constructs that this lame converter does
    #not understand.  Solution, make and translate from a bogus
    #test_equipment.bbx

10   if ($include_file =~ /^test_equipment\.v/)
    {
        &Make_Verilog_Test_BBX();
        $include_file = "test_equipment.bbx";
15   }
    next if ($include_file =~ /^modelsim_define\.v.*$/);

    push (@all_sim_files,$include_file);
}

20   push (@all_sim_files,$file);

    my @output_files = &V2VHD_Files("`define SIMULATION",".",@all_sim_files);
    &Make_VHDL_Test_Equipment;

25   $" = "\n";
    print "Here is a list of all 1993 vhdl files generated by this script:\n
@output_files\n";

30   #print "vcom -93 -explicit @output_files\n";
}
sub Check_For_Altera_Copyright
{
    my $VERILOG_COPYRIGHT_NOTICE=
35   qq[//Copyright (C) 1991-2001 Altera Corporation
    //Any megafunction design, and related net list (encrypted or decrypted),
    //support information, device programming or simulation file, and any other
    //associated documentation or information provided by Altera or a partner
    //under Altera's Megafunction Partnership Program may be used only to
40   //program PLD devices (but not masked PLD devices) from Altera.  Any other
    //use of such megafunction design, net list, support information, device
    //programming or simulation file, or any other related documentation or
    //information is prohibited for any other purpose, including, but not
45   //limited to modification, reverse engineering, de-compiling, or use with
    //any other silicon devices, unless such use is explicitly licensed under
    //a separate agreement with Altera or a megafunction partner.  Title to
    //the intellectual property, including patents, copyrights, trademarks,
    //trade secrets, or maskworks, embodied in any such megafunction design,
    //net list, support information, device programming or simulation file, or
50   //any other related documentation or information provided by Altera or a
    //megafunction partner, remains with Altera, the megafunction partner, or
    //their respective licensors.  No other licenses, including any licenses
    //needed under any third party's intellectual property, are provided herein.
    //Copying or modifying any file, or portion thereof, to which this notice
55   //is attached violates this copyright.];

    my @Altera_Files;
    foreach $file (@_)
    {
60        my $fs = HDL_Read_File($file);
        push (@Altera_Files, $file)
        ;#      if ($fs =~ /$VERILOG_COPYRIGHT_NOTICE/);
    }
}

```

TOTAL 90" GOT 08860

```

    return (@Altera_Files);
}

sub HDL_Read_File
5  {
    my $file = shift or die "no file specified\n";
    open (FILE, "< $file") or die "cannot open file ($file) ($!)\n";
    my $return_string;
    while (<FILE>)
10  {
        $return_string .= $_;
    }
    close (FILE);
    return($return_string);
15 }

sub HDL_Remove_Comments
{
    my $string = shift or die "no string specified\n";
    my $language = shift || "verilog";
20
    if ($language =~ /verilog/i)
    {
        $string =~ s|\\\/\*.*?\*\\\/||gs;
        $string =~ s|\\\/\*.*$||gm;
        return ($string);
25    }
    if ($language =~ /vhdl/i)
    {
        $string =~ s|\\\/\*.*$||gm;
        return ($string);
30    }
    die "HRC, language ($language) not understood\n";
}
35

sub Make_Verilog_Test_BBX
{
    my $string;
    #first read in verilog test equipment and save it
    #away for later.
    open (FILE, "< test_equipment.v") or
40    die "cannot open test_equipment.v ($!)\n";
    while (<FILE>)
    {
        $string .= $_;
45    }
    close (FILE);

    open (FILE, "> test_equipment.bbx") or
50    die "cannot open test_equipment.v ($!)\n";
    print FILE <<END_OF_TEST_EQUIPMENT_V;
    //////////////////////////////////////
    // MODULE: Clk_Gen
    //
55    // Test Equipment.
    //
    // Generates a 50%-duty-cycle continuous clock. Its one parameter
    // sets the period.
    //
60    //////////////////////////////////////
    module Clk_Gen (clk);
        output clk;
        //put something in to keep from being declared a black box

```

```

    wire    foo = 1;
endmodule // Clk_Gen

```

```

5  ///////////////////////////////////////////////////////////////////
   // MODULE: Reset_Gen
   //
   // Test Equipment.
   //
10  // Generates a reset pulse (logic negative)
   // 100 clock-periods long at the beginning of the
   // simulation.
   //
   ///////////////////////////////////////////////////////////////////
15  module Reset_Gen (reset_n);
      output reset_n;

      //put something in to keep from being declared a black box
      wire    foo = 1;

20  endmodule // Reset_Gen
   END_OF_TEST_EQUIPMENT_V
      close (FILE);

25  return ($string);
   }
   sub Make_VHDL_Test_Equipment
   {
30     my $file = "test_equipment.vhd";
       open (FILEOUT,">$file") or die "cannot open file ($file) ($!)\n";
       print FILEOUT <<END_OF_TEST_EQUIPMENT;

       --Copyright (C) 1991-2000 Altera Corporation
       --Any megafunction design, and related net list (encrypted or decrypted),
35  --support information, device programming or simulation file, and any other
       --associated documentation or information provided by Altera or a partner
       --under Altera's Megafunction Partnership Program may be used only to
       --program PLD devices (but not masked PLD devices) from Altera. Any other
       --use of such megafunction design, net list, support information, device
40  --programming or simulation file, or any other related documentation or
       --information is prohibited for any other purpose, including, but not
       --limited to modification, reverse engineering, de-compiling, or use with
       --any other silicon devices, unless such use is explicitly licensed under
       --a separate agreement with Altera or a megafunction partner. Title to
45  --the intellectual property, including patents, copyrights, trademarks,
       --trade secrets, or maskworks, embodied in any such megafunction design,
       --net list, support information, device programming or simulation file, or
       --any other related documentation or information provided by Altera or a
       --megafunction partner, remains with Altera, the megafunction partner, or
50  --their respective licensors. No other licenses, including any licenses
       --needed under any third party's intellectual property, are provided herein.

       LIBRARY ieee;
       use ieee.std_logic_1164.all;
55  use ieee.std_logic_arith.all;
       use ieee.std_logic_unsigned.all;

       ENTITY Clk_Gen IS
           PORT (
60             SIGNAL clk : OUT STD_LOGIC
           );
       END Clk_Gen;

```

09880106-061201

```

LIBRARY ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

5  ENTITY Reset_Gen IS
    PORT (
        SIGNAL reset_n : OUT STD_LOGIC
    );
10  END Reset_Gen;

LIBRARY ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
15  use ieee.std_logic_unsigned.all;

ARCHITECTURE behavior OF Clk_Gen IS
    SIGNAL tmp_clk : STD_LOGIC_VECTOR(0 DOWNTO 0);
BEGIN
20  process (tmp_clk)
begin -- process
    if tmp_clk = "0" then
        tmp_clk <= "1" after 15 ns, "0" after 30 ns;
25  else
        tmp_clk <= "0" after 15 ns;
    end if;
    end process;
    clk <= tmp_clk(0);
30  END behavior;

ARCHITECTURE behavior OF Reset_Gen IS
    SIGNAL tmp_reset_n : STD_LOGIC_VECTOR(0 DOWNTO 0);
BEGIN
35  PROCESS
    BEGIN

        tmp_reset_n <= "0" ;
40        wait for 3000 ns;
        tmp_reset_n <= "1" ;
        WAIT;
    END PROCESS;
    reset_n <= tmp_reset_n(0);
45  END behavior;
END_OF_TEST_EQUIPMENT

    close (FILEOUT);
}
50  sub Replace_Cpu_Register_Ram
{
    my $memory_signal_name = shift or die "RCRR, no sig given";
    return <<END_OF_REGISTER_RAM;
55  PROCESS(clk,rdaddress)
    BEGIN
        q <= $memory_signal_name\ (CONV_INTEGER(unsigned(rdaddress))\);
        if rising_edge(clk) then
            last_wraddress <= wraddress;
60        IF wren /= '0' THEN
            $memory_signal_name\ (CONV_INTEGER(unsigned(last_wraddress))\ ) <=
data;
                IF (last_wraddress = rdaddress) THEN

```

```
q <= data;
END IF;
end if;
END IF;
5  END PROCESS;
END_OF_REGISTER_RAM
}
```

09220106-061201
T02T90-90T03260

Crush_names.pm

```
#####
# Crush_Names_That_Are_Bigger_Than_Max_Width_Characters
5 #
# One of the great benefits of using a higher programming language
# is the ability to assign descriptive variable names, such that a
# human of modest programming ability may peruse the code and
# comprehend its workings.
10 #
# There seem to be certain programmers in this world who do not
# believe that description can be expressed beyond a certain number
# of characters. Some unnamed synthesis tools, for instance, lob off
# all characters beyond the 32nd, or modify these character-
15 # heavy names in their own way, or simply implode in a screaming
# error message of death.
#
# Unfortunately, we are a long-fingered bunch who love to hear the
# sounds of our own typing. We have many Verilog/VHDL wires, regs and
20 # filenames that regularly exceed smaller limitations.
# Furthermore, many names use other objects' names as prefixes and
# suffixes, pushing the character-count even higher.
#
# This long-named subroutine, ironically, "crushes names that
25 # are bigger than a maximum width of characters" to accomodate
# such character-width-limited programs.
#
# All-in-all, I do not begrudge the character-truncating programs
# for their efforts. After all, they (usually) produce working
30 # synthesized output-files... with one exception. Often, these
# programs will rename a module without renaming the filename where
# this module resides ... which is one of the attributes we rely
# upon for Quartus to find black boxes in the synthesized code.
# Thus, we crush not only names in the code, but filenames as well.
35 #
# This subroutine must then not only modify the contents of files,
# but regenerate the list of files with named-crushed filenames and
# modify the hash that translates the original-name to the crushed-name.
#
40 #####
sub Crush_Names_That_Are_Bigger_Than_Max_Width_Characters
(
    my ($pConverted_Filenames,
        $pConversion_Hash,
45         $max_width,
        @File_List) = @_;
    my $max_width_minus_1 = $max_width - 1;
    my $max_width_plus_1 = $max_width + 1;

50     my @new_File_List;
    my $line;
    #my %Reverse_Hash = reverse %$pConversion_Hash;

    foreach $filename (@File_List)
55     (
        $filename =~ tr|\\|\\/|;
        my $new_filename;

        # test to see if we've already converted this file
60         $new_filename=$pConverted_Filenames{$filename};
        if ($new_filename ne "") {
            push (@new_File_List, $new_filename);
            next;
        }
    )
}
```

0380106-061201


```

)

#####
# A filename bigger than $max_width characters often refers to a module
# that is bigger than $max_width characters.
# Rename the filename to a $max_width character filename.
$new_filename = $filename;
if ($new_filename =~ s/(\w{$max_width,})(\.[^\./]*)$/
    &Make_Max_Width_Char_Name($pConversion_Hash,$max_width,$1).
    $2/sgoex)
{
    if ($filename ne $new_filename)
    {
        rename $filename, $new_filename or
        die "ERROR Crush_Names_....: could not rename file ($filename)
to ($new_filename) ($!)\n";

        if (!( -T $new_filename ))
        {
            rename $filename.$suffix, $new_filename.$suffix or
            die "ERROR Crush_Names_....: could not rename file
($filename$suffix) to ($new_filename$suffix) ($!)\n";
        }
    }
}

# Whether we rename the file the file or not, we then need to
# add filename into others;
push (@new_File_List, $new_filename);
$$pConverted_Filenames{$filename}=$new_filename;

$new_filename_for_reading = $new_filename;
$do_AF=0;
if (!( -T $new_filename )) {
    $new_filename_for_reading .= $suffix;
    $do_AF=1;
}
open (FILE, "< $new_filename_for_reading") ||
    die "ERROR Crush_Names_....: CANNOT OPEN FILENAME FOR
40 READING($new_filename_for_reading) ($!)\n";
my $output_string = "";
while ($line = <FILE>)
{
    $output_string .= &Crush_Line ($line,$max_width,$pConversion_Hash);
45 }
close (FILE);

if ($do_AF) {
    open (ALTERA_FILEHANDLE, "> $new_filename") ||
    die "ERROR Crush_Names_....: CANNOT OPEN FILENAME FOR
50 WRITING($new_filename) ($!)\n";
    print ALTERA_FILEHANDLE $output_string;
    close (ALTERA_FILEHANDLE);
} else {
    open (FILE, "> $new_filename") ||
    die "ERROR Crush_Names_....: CANNOT OPEN FILENAME FOR
55 WRITING($new_filename) ($!)\n";
    print FILE $output_string;
    close (FILE);
60 }
} # end of foreach

return (@new_File_List);

```

0920106-061201
102190-9003366

09880106-061201
T02T90-90T08860

```

}

#####
#
5 sub Crush_Line
{
    my ($line,$max_width,$pConversion_Hash) = @_;
    my $max_width_minus_1 = $max_width - 1;
    if ($line =~ s/([a-zA-
10 Z\_]\w{$max_width_minus_1,})/&Make_Max_Width_Char_Name($pConversion_Hash,$max_w
    idth,$1)/sgoe)
    {
        #print "Crush_Names... : line was ($old_line) now ($line)\n";
    }
    return ($line);
15 }

#####
sub Make_Max_Width_Char_Name
20 {
    my ($pConversion_Hash,$max_width,$word) = @_;
    my %Reverse_Hash = reverse %$pConversion_Hash;

    # first, check to see if word has appeared before
    my $new_word = $$pConversion_Hash{$word};
    if ($new_word eq "")
    {
        $new_word = $word;

        # word has not been found before, and we need to make a new hash entry
    30 my $max_width_minus_1 = $max_width - 1;
        my $max_width_plus_1 = $max_width + 1;
        die "ERROR Make_Max_Width_Char_Name: WORD ($new_word) not understood\n"
            unless ($new_word =~ s/^([a-zA-
35 Z\_]\w{0,$max_width_minus_1})(\w*)/$1/);
        # (also, crushes it down to $max_width charactes)

        #If word ends with \_ vhdl hates you
        #so fool while loop by assigning value to reverse hash
    40 if ($new_word =~ /\_$/ )
        {
            $Reverse_Hash{$new_word} = "vhdl placeholder";
        }

    45 while ($Reverse_Hash{$new_word})
        {
            #strip off the number at the end of the 32 char word
            #and increment it.
            if ($new_word =~ /\^(\\w*?)(\\d+)$/)
            50 {
                my $base = $1;
                my $index = $2 + 1;
                while (($base.$index) =~ /\(\\w{$max_width_plus_1,})/o)
                {
                    chop ($base);
                }
                $new_word = $base.$index;
                #print " word is numbered ($new_word)\n";
            }
            else
            60 {
                $new_word =~ s/\^(\\w{$max_width_minus_1}).*$/$1/o;
                $new_word .= "1";
            }
        }
    }
}

```

00000106 0612001
T02150" 90T08860

```
        #print " word was unnumbered ($new_word)\n";
    }
    # check to make sure we're not outta hashes
5    die "ERROR Make_Max_Width_Char_Name: WORD ($new_word) not understood.
Out of hashes.\n"
        unless ($new_word =~ /^[a-zA-Z_](1)\w{0,$max_width_minus_1}$/);
        $$pConversion_Hash{$word} = $new_word;
    }
10    return ($new_word);
}

sub Make_MIF_Files_Max_Friendly
{
15    my ($pConverted_Filenames,
        $pConversion_Hash,
        $max_width,
        @File_List) = @_;

20    my @converted_file_list =
        &Crush_Names_That_Are_Bigger_Than_Max_Width_Characters (@_);

    foreach $file (@converted_file_list)
    {
25        open (MIFFILE,"< $file") or
            die "Cannot open mif file ($file) ($!)\n";
        my $file_content;
        while (<MIFFILE>){$file_content .= $_;}
        close (MIFFILE);
30        $file_content =~ s|\\/*|\\%|g; #max+2 likes % instead of /* */
        $file_content =~ s|\\*\\/|\\%|g; #max+2 likes % instead of /* */

        $file_content =~ s|\\-\\-.*$||mg; #single comments mp2 hates them
        $file_content =~ s|\\bUNS\\b|DEC|g; #change type UNS to DEC

35        open (MIFOUT,"> $file") or
            die "Cannot open mif file ($file) ($!)\n";
        print MIFOUT $file_content;
        close (MIFOUT);
40    }
    return (@converted_file_list);
}

45    1; # Modules must say "1"--mustn't they?
```

default_generator_program.pl

```
#####
# default_generator_program.pl
#
# Suppose a user has a plain-old Verilog (or VHDL) module--no
# Vpp, parameterization, or generation about it.
#
# Suppose they want to include this design into an Avalon system
# module. They are perfectly willing to fill out a "class.ptf" file
# which describes their ports and sundry System-parameters
# (e.g. Hold_Time). But they don't want to write a whole generator
# program--they just want to use their "thing".
#
# If the component's class.ptf file gives "" (or "--default--") as
# the "generator_program", then the SOPC-Builder calls THIS VERY
# PROGRAM (default_generator_program.pl) -as if- it was the user's
# generator program. This saves the Authors of simple
# SOPC-components the drudgery of writing their own "generator_program."
#
# And what wonderful things do we do on the author's behalf?
#
# 1) Generate a renaming-wrapper.
# 2) Copy implementation-files into project-directory.
# 3) Arrange for some files to be synthesized (if appropriate or asked).
#
# **** Making a Renaming Wrapper
#
# The user will have defined a module named, for example,
# jthingwizard (who knows--they -might- name it that!).
# But, in the SOPC-Builder table, they want to add
# one of those "jthingwizard" things named "my_inst."
# So we create a new module named "my_inst" (with
# exactly the same ports as "jthingwizard") which contains
# one instance of "jthingwizard" and nothing else.
# The almost-pointless (but totally required) "my_inst"
# module is the -renaming wrapper-.
#
# When we build the renaming-wrapper, we can either instantiate the
# user's module directly, or we can instantiate it as a black-box.
# This all depends on whether they want us to synthesize it or not.
# often they will, but sometimes they won't. If you don't explicitly
# say so, this program will try to make an intelligent guess about
# black-boxing based on the types of files found in the class-directory.
# If you have nothing but a schematic, for example, then clearly you
# want a black-box.
#
# It is only possible to build a renaming wrapper if:
# a) The users' "class.ptf" file describes -all- ports.
# b) The top-level module name is known.
#
# (a) is entirely up to you. If you don't describe each-and-every
# port in your "class.ptf"-file, then this function simply will not
# work.
#
# We try to help you with (b) by making an educated guess. If you
# don't specifically say what your top-level module is, we just
# assume it's the same as the class-name (what else would it be,
# after all?).
#
# **** Copying Implementation Files
#
```

09880106 061201

```

# Every component must be implemented -somehow-. We start off with
# the default assumption that the top-module of this component is
# named the same as the component directory ("jthingwizard" in the
# above example) and is implemented in an HDL, BDF, or EDF-file of
5 # the same name. Unless told otherwise, we search for such a file in
# the components' directory, and then copy it blindly into the
# project directory for the system-under-construction. In fact,
# by default, we copy -all- design files from the component-directory into
# the project-directory, whether you need them or not (if you didn't
10 # need them, after all, what were they doing in the component directory?).
#
# Alternatively, the user can supply a list of files-to-be copied
# and, separately, files-to-be-synthesized.
#
15 # **** Altering default behavior.
#
# Suppose the top-level module -isn't- the same as the
# component-directory name, or -isn't- implemented in a file
# of the same name. Or, suppose you -don't- want us to try
20 # synthesizing your file, but you want it to be a black-box instead.
# well, then. You'd have to override the behavior of this program
# somehow. (You could, of course, write your own generator program,
# but we'll see how far we can get you before we ask you to bail
# out).
25 #
# Your "class.ptf" file can contain a section named
# DEFAULT_GENERATOR, in which you can supply directions to this
# program. Here is an example DEFAULT_GENERATOR function
# demonstrating all the available parameters:
30 #
#     DEFAULT_GENERATOR
#     {
#         top_module_name      = "my_tippie_toppie";
#         black_box_files      = "summit.edf, blickman.bsf";
#         synthesis_files      = "toppo.v, submod1.v, libx.vhd";
35 #         black_box          = "0";
#     }
#
# Seems pretty clear to me what all these things are, so I'll spare
40 # you the pain of suffering through an English description.
#
# All the filenames are given relative to the class-directory
# (or as absolute full pathnames, your choice).
#
45 #####
use wiz_utils;

#####
50 # List_Ports_From_PTF
#
# Given a ptfRef which contains a "MODULE" section
# we should be able to do a good-old VPP "&List_Ports_For" call.
#
55 # This function does exactly that.
#
#####
sub List_Ports_From_PTF
(
60     my ($db_Module) = (@_);

    my $module_name = &get_data ($db_Module);
    my $db_Wiring   = &PTF_Get_Required_Child_By_Path($db_Module, "PORT_WIRING");

```

```

my @port_description_list = ();

my $num_children = &get_child_count ($db_Wiring);
5  for (my $child_index=0; $child_index < $num_children; $child_index++) {
    my $db_Port = &get_child($db_Wiring, $child_index);
    next unless &get_name($db_Port) eq "PORT";

    my $Port = &PTF_Build_Hash_From_Section ($db_Port);

10    my @attribute_list = ("name=" . &get_data($db_Port));
    foreach $attr (keys(%$Port)) {
        push (@attribute_list, "$attr = $$Port{$attr}");
    }
15    push (@port_description_list, join " | ", @attribute_list);
}

&List_Ports_For ($module_name, @port_description_list);
}

20 #####
# Default_Generator
#
# Implements the default generator as a callable function.
#
25 #####
sub Default_Generator
{
    my ($arg, $user_defined, $db_Module, $db_PTF_File) =
30    &Process_Wizard_Script_Arguments ("", @_);

    &Progress ("Default Generator Program for: $$arg{name}.");

    # Open-up "class.ptf" and extract DEFAULT_GENERATOR, PORT_WIRING
35    my $db_Class_File =
        &PTF_New_Required_Ptf_From_File ("$$arg{class_directory}/class.ptf",
            "No 'class.ptf' file found for module $$arg{name}");

    my $class_name = &PTF_Get_Required_Data_By_Path ($db_Module, "class");
40    my $gen_args = &PTF_Build_Hash_From_Section ($db_Class_File,
        "CLASS/DEFAULT_GENERATOR",
        "0"); # OK if section missing.

45    if ($$gen_args {black_box} eq "") {
        # Try to guess whether they want their design synthesized, or
        # included as a black-box (since they didn't say).
        #
        # BAD Heuristic:
50    #   If there are any HDL-files at all, then we'll try to synthesize
        #   them.
        #
        # if (scalar (@synth_full_paths) == 0) {
        #   &Progress("Default Generator: treating $$arg{name} as a black-box");
        #   $$gen_args{black_box} = 1;
55    # } else {
        #   &Progress("Default Generator: ($$arg{name}) adding files for synthesis");
        #   $$gen_args{black_box} = 0;
        # }
60    # GOOD Heuristic:
        #   Only synthesize if they explicitly set black_box = 0.
        #
        $$gen_args {black_box} = 1;

```

09380106 061201

```

}

#####
#time to update the system port_wiring.
5 &delete_child($db_Module,"PORT_WIRING");
my $db_ports = &get_child_by_path($db_Class_File,
                                "CLASS/MODULE_DEFAULTS/PORT_WIRING");
&add_child($db_Module,"",$db_ports);
#####
10 # First, go through the files in the directory. This will give us
# a clue about whether to treat the users' design as a black-box
# or as a synthesizable thing.
#
opendir (CLASSDIR, "$$arg{class_directory}");
15
my @bb_base_files = split (/\\s*\\,\\s*/, $$gen_args{black_box_files});
if (scalar (@bb_base_files) == 0) {
    foreach $file (readdir(CLASSDIR)) {
        next unless $file =~ /\. (tdf|edf|bsf|vqm|mif)$/;
20         push (@bb_base_files, "$file");
    }
}

my @synth_base_files = split (/\\s*\\,\\s*/, $$gen_args{synthesis_files});
25 if ((scalar (@synth_base_files) == 0) &&
    (!$$gen_args{black_box} ) ){
    rewinddir (CLASSDIR);
    foreach $file (readdir(CLASSDIR)) {
        next unless $file =~ /\. (v|vhd)$/;
30         push (@synth_base_files, "$file");
    }
}

closedir (CLASSDIR);
35
# Make all files relative to the class_directory (unless, of course,
# the path is already absolute).
#
my @bb_full_paths = ();
40 my @synth_full_paths = ();
my @synth_proj_files = (); #as they appear after copying into project.

foreach $bb_file (@bb_base_files) {
    $bb_file = "$$arg{class_directory}/$bb_file"
45     if !&Is_Absolute_Path ($bb_file);
    push (@bb_full_paths, $bb_file);
}

foreach $synth_file (@synth_base_files) {
50     $synth_file = "$$arg{class_directory}/$synth_file"
    if !&Is_Absolute_Path ($synth_file);
    push (@synth_full_paths, $synth_file);

    push (@synth_proj_files,
55         "$$arg{system_directory}/".&Get_Base_Fname($synth_file));
}

if (scalar(@synth_full_paths) + scalar(@bb_full_paths) == 0) {
    print STDERR "WARNING: Default Generator Program (run for $$arg{name})\n";
60     print STDERR "          No design-files found. Probably not good.\n";
}

# Copy the files:

```

00000106"061201

```
foreach $srcfile (@bb_full_paths, @synth_full_paths) {
    &Perlcopy ($srcfile, "$$arg{system_directory}");
}
```

```

5  $$gen_args {top_module_name} = $class_name
   if $$gen_args {top_module_name} eq "";

#####
10 # Build the renaming wrapper
   #
   my $wrapper_name = &get_data ($db_Module);
   my $wrapper_file = "$$arg{system_directory}/$wrapper_name.v";
   my $instance_name = "the_$$gen_args{top_module_name}";

15 # Make Vpp port-lists for both the wrapper and the wrappee
   # (they happen to have exactly the same ports).
   #
   &List_Ports_From_PTF ($db_Module);
20 my $port_descriptions = &Describe_Ports_For ($wrapper_name);
   &List_Ports_For ($$gen_args{top_module_name}, $port_descriptions);

   my $port_defs = &Define_Ports_For ($wrapper_name, 1);
   my $port_decs = &Declare_Ports_For ($wrapper_name, 1);

25 # A bunch of here-strings containing chunks of the wrapper-file:
   #
   my $top_comment=<<EOMC;
   //
30 // Renaming wrapper
   // built by 'default_generator_program.pl'.
   // Wraps 'simple' component class $class_name,
   // implemented by the top-level module
   // $$gen_args{top_module_name}.

35 EOMC

   my $black_box_declaration=<<EOM1;
   // synopsys translate_off
40 `ifdef LEONARDO_SPECTRUM
   // synopsys translate_on
   // Black-box declaration for simple module $$gen_args{top_module_name}
   module $$gen_args{top_module_name} (
       $port_decs
45 )/* synthesis syn_black_box */;
       $port_defs
   endmodule
   // synopsys translate_off
   `endif
50 // synopsys translate_on
   EOM1

   my $wrapper_declaration=<<EOM2;

55 module $wrapper_name (
       $port_decs
   );
       $port_defs
   EOM2

60 open (ALTERA_FILEHANDLE, "> $wrapper_file") or die
   "Default_Generator: couldn't open wrapper file $wrapper_file: $!";
   my $old_out = select (ALTERA_FILEHANDLE);

```


09880106.06.120.1
Total 90" 90T0886

```
print ALTERA_FILEHANDLE $GLOBAL_COPYRIGHT_NOTICE;
print ALTERA_FILEHANDLE $top_comment;
print ALTERA_FILEHANDLE $black_box_declaration if ($$gen_args{black_box});
5  print ALTERA_FILEHANDLE $wrapper_declaration;

print ALTERA_FILEHANDLE &Instantiate_And_Connect_Statement
    ($$gen_args{top_module_name}, $instance_name);

10  print ALTERA_FILEHANDLE "// exemplar attribute $instance_name NOOPT TRUE\n"
    if ($$gen_args{black_box});
    print ALTERA_FILEHANDLE "endmodule\n";

    close (ALTERA_FILEHANDLE);
15  select ($old_out);

    push (@synth_proj_files, $wrapper_file);

    # Add to synthesis HDL-file list:
20  &Add_Synthesis_Files_To_PTF ($db_Module, @synth_proj_files);
    &write_ptf_file ($db_PTF_File) or die
        "Couldn't write PTF File when generating $$arg{name}";

    }
25  #####
    #####
    ###
    ###      Execution starts here.
30  ###
    #####
    #####

    &Default_Generator (@ARGV);
35

40
```

mk_bsf.pm

```
5  #Copyright (C) 1991-2001 Altera Corporation
   #Any megafunction design, and related net list (encrypted or decrypted),
   #support information, device programming or simulation file, and any other
   #associated documentation or information provided by Altera or a partner
   #under Altera's Megafunction Partnership Program may be used only to
   #program PLD devices (but not masked PLD devices) from Altera. Any other
10  #use of such megafunction design, net list, support information, device
   #programming or simulation file, or any other related documentation or
   #information is prohibited for any other purpose, including, but not
   #limited to modification, reverse engineering, de-compiling, or use with
   #any other silicon devices, unless such use is explicitly licensed under
15  #a separate agreement with Altera or a megafunction partner. Title to
   #the intellectual property, including patents, copyrights, trademarks,
   #trade secrets, or maskworks, embodied in any such megafunction design,
   #net list, support information, device programming or simulation file, or
   #any other related documentation or information provided by Altera or a
20  #megafunction partner, remains with Altera, the megafunction partner, or
   #their respective licensors. No other licenses, including any licenses
   #needed under any third party's intellectual property, are provided herein.
   #Copying or modifying any file, or portion thereof, to which this notice
   #is attached violates this copyright.

25  # Constants for computing dimensions of things.
   # distance between inner and outer rectangles, x and y
   my $outerEdgeMargin = 16;

   # Margin between top or bottom signal and inner rectangle.
30  my $innerEdgeMargin = 16;

   # x distance between longest left and longest right signal.
   my $defaultFontSize = 8;
   my $titleFontSize = 10;
35  my $quantum = $defaultFontSize;
   my $midMargin = 4 * $quantum;

   # Vertical space occupied by a signal.
40  my $signalVerticalSize = 16;

   my $defaultFont = "Arial";

   # Inter-module separator. When the symbol is rendered, the magic
   # string "[]" (unlikely to be a signal name) will become a dashed line.
45  my $secretDashedLineMarker = "[]";

   my $globalPrintString;

   sub myErrorPrint
50  {
     # print STDERR @_;
     warn @_;
   }

   sub myPrint
55  {
     # print @_;
     $globalPrintString .= join ' ', @_;
   }

60  sub charWidth
   {
     return charDim("width", @_);
   }
```

09880106-061201
TOTAL 50 " 90108866

TOTAL 90" 90T08860

```

}
sub charDimArial
{
5   my ($axis, $char, $size) = @_;

   if ($size != 8 && $size != 10)
   {
10    myErrorPrint "Unsupported font size $size, using $defaultFontSize.\n";
    $size = $defaultFontSize;
   }

   # Size data was taken from Quartus, at zoom level 144%.
   my %size8ToDim = (
15    'a' => 248/40.0,
    'b' => 248/40.0,
    'c' => 224/40.0,
    'd' => 248/40.0,
20    'e' => 248/40.0,
    'f' => 112/40.0,
    'g' => 248/40.0,
    'h' => 224/40.0,
    'i' => 112/40.0,
25    'j' => 84/40.0,
    'k' => 224/40.0,
    'l' => 84/40.0,
    'm' => 360/40.0,
    'n' => 224/40.0,
30    'o' => 248/40.0,
    'p' => 248/40.0,
    'q' => 248/40.0,
    'r' => 136/40.0,
    's' => 224/40.0,
    't' => 112/40.0,
35    'u' => 224/40.0,
    'v' => 184/40.0,
    'w' => 304/40.0,
    'x' => 192/40.0,
    'y' => 192/40.0,
40    'z' => 192/40.0,
    'A' => 304/40.0,
    'B' => 304/40.0,
    'C' => 336/40.0,
    'D' => 336/40.0,
45    'E' => 304/40.0,
    'F' => 276/40.0,
    'G' => 336/40.0,
    'H' => 304/40.0,
    'I' => 84/40.0,
50    'J' => 224/40.0,
    'K' => 304/40.0,
    'L' => 248/40.0,
    'M' => 360/40.0,
    'N' => 304/40.0,
55    'O' => 336/40.0,
    'P' => 304/40.0,
    'Q' => 336/40.0,
    'R' => 304/40.0,
    'S' => 304/40.0,
60    'T' => 248/40.0,
    'U' => 304/40.0,
    'V' => 304/40.0,
    'W' => 416/40.0,

```

```

'x' => 304/40.0,
'y' => 248/40.0,
'z' => 248/40.0,
'0' => 248/40.0,
'1' => 248/40.0,
'2' => 248/40.0,
'3' => 248/40.0,
'4' => 248/40.0,
'5' => 248/40.0,
'6' => 248/40.0,
'7' => 248/40.0,
'8' => 248/40.0,
'9' => 248/40.0,
'_' => 248/40.0,
'-' => 248/40.0,
'[' => 112/40.0,
']' => 112/40.0,
'.' => 112/40.0,
);

```

```

return ($size / 8) * $size8ToDim{$char} if (exists($size8ToDim{$char}));

```

```

myErrorPrint "Unsupported character! Font: Arial; char: '$char'; size:
$size\n";
return 248/40.0;
}

```

```

sub charDim
{

```

```

my ($axis, $char, $font, $size) = @_;
my %sizeToDimCourierNew = (
    8 => 6.7,
    9 => 7.06,
    10 => 8.11,
    11 => 9.09,
    12 => 10.2,
    14 => 11.1,
    16 => 13.3,
    18 => 14.3,
);

```

```

# For now, I assume Courier New.

```

```

if ($font eq "Arial")
{
    return charDimArial($axis, $char, $size);
}

```

```

if ($font ne $defaultFont)
{
    myErrorPrint "Unsupported font '$font', using $defaultFont.\n";
    $font = $defaultFont;
}

```

```

if (!exists($sizeToDimCourierNew{$size}))
{
    myErrorPrint "Unsupported font size $size, using $defaultFontSize.\n";
    $size = $defaultFontSize;
}

```

```

if ($axis eq "width")
{
    return $sizeToDimCourierNew{$size};
}

```

T02T90-90T08850

```

    }

    if ($axis eq "height")
    {
5      return $sizeToDimCourierNew($size);
    }

    myErrorPrint "Unsupported axis '$axis'\n";

10   return $sizeToDimCourierNew{$defaultFontSize}
}

sub stringWidth
{
15   my ($string, $font, $size) = @_;
    my $i;
    my $len = 0;

    for $i (0 .. length($string) - 1)
20   {
        $len += charWidth(substr($string, $i, 1), $font, $size);
    }

    return $len;
25 }

sub getMaxSignalNameWidth
{
    my @signals = @_;
30   my $signal;
    my $max = -1;

    foreach $signal (@signals)
    {
35       if ($signal =~ /
            (\S+)          # signal name, perhaps with a bus size element.
            \s*\|\s*       # pipe, maybe with whitespace
            \d+            # Bus width
            \s*\|\s*       # pipe, maybe with whitespace
40         \w+             # signal type
            /sx)
        {
            my $signalName = $1;
            my $len = stringWidth($signalName, $defaultFont, $defaultFontSize);
45             if ($max < $len)
                {
                    $max = $len;
                }
        }
50     }

    return $max;
}

55 sub max
{
    my ($a, $b) = @_;

    return $a if ($a > $b);
60   return $b;
}

sub roundUp

```

09220106-061201

```

{
    my ($num, $quantum) = @_;

    my $diff = $num / $quantum - int($num / $quantum);
5
    if (0 == $diff)
    {
        return $num;
    }

10
    $num = (int($num / $quantum) + 1) * $quantum;
    return $num;
}

15
sub translate
{
    my ($tX, $tY, $xMin, $yMin, $xMax, $yMax) = @_;

    return ($xMin + $tX, $yMin + $tY, $xMax + $tX, $yMax + $tY);
20
}

sub computeDimensions
{
    my ($title, $instanceName, $leftSignalRef, $rightSignalRef) = @_;

25
    my @leftSignals = @$leftSignalRef;
    my @rightSignals = @$rightSignalRef;

    #
30
    # Collect some numbers.
    #

    # A symbol is some rectangles, lines and strings. Spatial values of and
    between
35
    # the rectangles and lines are determined by the number and length of the
    strings.
    # Strings:
    # title
    # instanceName
    # leftSignals[]
    # rightSignals[]
40
    # rightMaxLen
    # signals

    # I have two concentric rectangles.
45
    # The width depends on the lengths of the signal names; the height depends on
    # the number of signals.

    # The inner rectangle's width is determined by the largest left and right
    # signal names. But, if the title or the instance name are absurdly large,
50
    # they can override.
    my $signal;
    my $leftMaxLen = getMaxSignalNameWidth(@leftSignals);
    my $rightMaxLen = getMaxSignalNameWidth(@rightSignals);

55
    my $innerXMax = $leftMaxLen + $rightMaxLen + $midMargin;
    $innerXMax = max($innerXMax, $quantum + stringWidth($title, $defaultFont,
    10));
    $innerXMax = max($innerXMax, $quantum + stringWidth($instanceName,
    $defaultFont, 10));

60
    # The inner rectangle's height depends on the number of signals.
    my $innerYMax = (0 + @leftSignals) * $signalVerticalSize + $innerEdgeMargin;

```

```

    $innerXMax = roundUp($innerXMax, $quantum);
    $innerYMax = roundUp($innerYMax, $quantum);

    my @innerRect = (0, 0, $innerXMax, $innerYMax);
5    @innerRect = translate($innerEdgeMargin, $innerEdgeMargin, @innerRect);

    my $outerXMax = $innerXMax + 2 * $outerEdgeMargin;
    my $outerYMax = $innerYMax + 2 * $outerEdgeMargin;

10    return (@innerRect, 0, 0, $outerXMax, $outerYMax);
}

sub rect2String
{
15    my @rect = @_;

    return sprintf("%d %d %d %d", @rect);
}

20    sub emitPort
    {
        # A port spec in a bsf file looks like this:
        # (port
25    object declaration - the
        # (pt 0 32)
        port's attachment point - the
        # (input)
        port type - the
30    # (text "clk" (rect 0 0 12 13)(font "Courier New" (font_size 8)))
        text for the port (this one's invisible)
        # (text "clk" (rect 24 25 36 36)(font "Courier New" (font_size 8))) - the
        text for the port again, but this one shows up
        # (line (pt 0 32)(pt 16 32)(line_width 1)) - a
35    line between inner and outer rectangles
        # )

        my ($x, $y, $side, $signalSpec) = @_;

40    my ($signalName, $busWidth, $signalType);
        my ($pt1String, $pt2String);

        if ($side eq "left")
        {
45            $pt1String = "$x $y";
            $pt2String = sprintf("%d %d", $x + $outerEdgeMargin, $y);
        }
        else
        {
50            $pt1String = sprintf("%d %d", $x - $outerEdgeMargin, $y);
            $pt2String = "$x $y";
        }

        if ($signalSpec and $signalSpec ne $secretDashedLineMarker)
55    {
            $signalSpec =~ /
                (\S+)          # signal name, perhaps with a bus size element.
                \s*\|\s*       # pipe, maybe with whitespace
                (\d+)          # bus width
                \s*\|\s*       # pipe, maybe with whitespace
60            (input|inout|output) # signal type
            $/sx;

```

```

$signalName = $1;
$busWidth = $2;
$signalType = $3;

```

```

5   # BSF files call 'inout' signals 'bidir'.
    if ($signalType eq "inout")
    {
        $signalType = "bidir";
    }

10  my $lineWidth = ($busWidth == 1) ? 1 : 3;

    my @rect1 = (0, 0, stringWidth($signalName, $defaultFont,
$defaultFontSize), $signalVerticalSize);
15  my $rect1String = rect2String(@rect1);

    my @rect2;
    if ($side eq "left")
    {
20      @rect2 = translate($x + $outerEdgeMargin + 0.25 * $innerEdgeMargin, $y -
7, @rect1);
    }
    else
    {
25      @rect2 =
        translate($x - ($outerEdgeMargin + 0.65 * $innerEdgeMargin) -
stringWidth($signalName, $defaultFont, $defaultFontSize),
        $y - 7, @rect1);
    }
30  my $rect2String = rect2String(@rect2);

    # I've added a single space after signal names, to try to discourage
    # Quartus from truncating the last bits. I hope this doesn't break
    anything.
35  myPrint <<EOT;
    (port
    (pt $x $y)
    ($signalType)
    (text "$signalName " (rect $rect1String)(font "$defaultFont" (font_size
40  $defaultFontSize)))
    (text "$signalName " (rect $rect2String)(font "$defaultFont" (font_size
$defaultFontSize)))
    (line (pt $pt1String)(pt $pt2String)(line_width $lineWidth))
    )
45  EOT
    }
}

sub drawBSF
50 {
    my ($title, $instanceName,
        $leftSignalRef, $rightSignalRef,
        $innerXMin, $innerYMin, $innerXMax, $innerYMax,
        $outerXMin, $outerYMin, $outerXMax, $outerYMax) = @_;
55
    my @leftSignals = @$leftSignalRef;
    my @rightSignals = @$rightSignalRef;

    my @innerRect = ($innerXMin, $innerYMin, $innerXMax, $innerYMax);
60  my @outerRect = ($outerXMin, $outerYMin, $outerXMax, $outerYMax);

    my @titleRect = (0, 0, $quantum + stringWidth($title, $defaultFont,
$titleFontSize),

```

FOOTNOTES: 061201


```

    $signalVerticalSize);

    @titleRect = translate($quantum / 2, 0, @titleRect);

5    my $titleRectString = rect2String(@titleRect);

    my @instRect = (0, 0, $quantum + stringWidth($instanceName, $defaultFont,
$defaultFontSize),
    $signalVerticalSize);

10    @instRect = translate($quantum / 2, $outerYMax - 2 * $quantum, @instRect);

    my $instanceRectString = rect2String(@instRect);

15    my $innerRectString = rect2String(@innerRect);
    my $outerRectString = rect2String(@outerRect);

    myPrint <<EOT;
    (header "symbol" (version "1.1"))
20    (symbol
    (rect $outerRectString)
    (text "$title" (rect $titleRectString) (font "$defaultFont" (font_size
$titleFontSize)))
    (text "$instanceName" (rect $instanceRectString) (font "$defaultFont")))
25    EOT

    # Emit all the ports.
    my $signalSpec;
    my $i;
30    my $y = $outerEdgeMargin + $innerEdgeMargin;

    my $x = 0;
    my @dashedLineYs;

35    foreach $signalSpec (@leftSignals)
    {
        emitPort($x, $y, "left", $signalSpec);

        # Add this y value to the list of dashed lines to be drawn later.
40        if ($signalSpec eq $secretDashedLineMarker)
        {
            push @dashedLineYs, $y;
        }

45        $y += $signalVerticalSize;
    }

    $y = $outerEdgeMargin + $innerEdgeMargin;
    $x = $outerRect[2];
50    foreach $signalSpec (@rightSignals)
    {
        emitPort($x, $y, "right", $signalSpec);
        $y += $signalVerticalSize;
    }

55    myPrint <<EOT;
    (drawing
    EOT

60    foreach $y (@dashedLineYs)
    {
        my $endX = $innerRect[2] - 1;

```

```

myPrint qq{(line (pt $innerRect[0] $y) (pt $endX $y) (color 0 0
0) (dotted) (line_width 1))\n);
}

5 myPrint <<EOT;
  (rectangle (rect $innerRectString) (line_width 1)))
  )
  EOT
  }

10 sub Generate_BSF
  {
    my $title = shift;
    my $instanceName = "inst";

15    # Clear the global output string.
    $globalPrintString = "";

    # Split signals into left and right, and sort within modules.
20    my (@inputStrings) = @_;
    my @listOfModules;
    my $moduleSignalList;
    my @signals;

25    for $moduleSignalList (@inputStrings)
    {
      # Eliminate loathesome white space.
      $moduleSignalList =~ s/\s+//g;

30      @signals = split /,/, $moduleSignalList;
      while ($signals[$#signals] !~ /\S/)
      {
        pop(@signals);
      }

35      push @listOfModules, [ @signals ];
    }

    # Separate signals into inputs, outputs/inouts.
40    # txd_from_the_uarto | 1 | output, rxd_to_the_uarto | 1 | input
    #
    # Each string lists all the ports of a Nios submodule, comma-delimited.
    #
    # Separate the ports into (input) and (output, inout) and return as a list of
45    lists.
    # Use delimiters to make things line up nice.
    my @leftSignals;
    my @rightSignals;

50    # clk | 1 | input
    # txd_from_the_uarto | 1 | output
    # shared_data_bus_1_data | 16 | inout

    my $moduleRef;
55    my $i;
    for $i (0 .. $#listOfModules)
    {
      my $signalSpec;
      my @tmpLeft = ();
      my @tmpRight = ();
60      my $moduleSignalList = $listOfModules[$i];
      foreach $signalSpec (@$moduleSignalList)
      {

```

05880105-051201

```

if ($signalSpec =~ /(\S+)\s*\|s*(\d+)\s*\|s*(\S+)/s)
{
    my $signalName = $1;
    if ($signalName =~ /[^\w\[\]\.]/)
    {
        goldfish("unexpected character(s) in signal name '$signalName'");
        next;
    }

    my $busWidth = $2;

    if ($busWidth =~ /^[^\d]/)
    {
        goldfish("unexpected character(s) in bus width '$busWidth'");
        next;
    }

    if ($busWidth < 1)
    {
        goldfish("bus width '$busWidth' is bogus.");
        next;
    }

    my $signalType = $3;

    # Fix up bus signals with their size ([n-1 .. 0]) here.
    if ($busWidth > 1)
    {
        if ($signalName !~ /\S+\[ (\d+) \. \. 0 \]/)
        {
            $signalName .= sprintf("[%d..0]", $busWidth - 1);
        }
    }

    if ($signalName =~ /\S+\[ (\d+) \. \. 0 \]/)
    {
        if ($1 != $busWidth - 1)
        {
            goldfish("signal $signalName doesn't match its own buswidth
($busWidth)");
        }
    }

    my $newSignalSpec = "$signalName|$busWidth|$signalType";
    if ($signalType eq "input")
    {
        push @tmpLeft, $newSignalSpec;
    }
    elsif ($signalType eq "inout" or $signalType eq "output")
    {
        push @tmpRight, $newSignalSpec;
    }
    else
    {
        goldfish("Unexpected signal type '$signalType' in signal spec
'$signalSpec'");
    }
}

# Sort signals within a module. This provides a consistent, though
# not necessarily optimal ordering.
@tmpLeft = sort(@tmpLeft);

```

09880106-061201

```

@tmpRight = sort(@tmpRight);

# Pad out the shorter of the two lists...
while ((0 + @tmpLeft) > (0 + @tmpRight))
5 {
    push (@tmpRight, "");
}

while ((0 + @tmpLeft) < (0 + @tmpRight))
10 {
    push (@tmpLeft, "");
}

# If this wasn't the last module, put an inter-module spacer.
15 if ($i < $#listOfModules)
{
    push (@tmpRight, $secretDashedLineMarker);
    push (@tmpLeft, $secretDashedLineMarker);
}

20 push @leftSignals, @tmpLeft;
push @rightSignals, @tmpRight;
}

25 # We should have ended up with left and right signal lists of the same size.
if (@leftSignals != @rightSignals)
{
    myErrorPrint "Say, left and right don't have the same size.\n";
}

30 # Do some arithmetic.
my ($innerXMin, $innerYMin, $innerXMax, $innerYMax, $outerXMin, $outerYMin,
$outterXMax, $outterYMax) =
    computeDimensions($title, $instanceName, \@leftSignals, \@rightSignals);

35 # Spew the rendering commands.
drawBSF($title, $instanceName,
    \@leftSignals, \@rightSignals,
    $innerXMin, $innerYMin, $innerXMax, $innerYMax,
40 $outerXMin, $outerYMin, $outerXMax, $outerYMax);

# Done!
return $globalPrintString;
}

45 qq(
1   When I am grown to man's estate
2   I shall be very proud and great,
3   And tell the other girls and boys
50 4   Not to meddle with my toys.
);

```

09080106 061201

```

mk_custom_sdk.pm

# use strict;      # please turn this off before checking in.

5  use ptf_parse;

my $gDebug = 0;
#
10 # 2000 August
#   dvb \ Altera Santa Cruz

#
# name of the directory within each wizard's dir
15 # with doc, inc, src, and lib files
#

my $CUSTOM_SDK_PIECES_DIR = "custom_sdk_pieces";

20 my $gUseOldSillyJNames = 0;  # if 1, convert to old juartwizard style names.

# -----
# bail(string)
#
25 sub bail
{
    my $x = shift;

    30     warn("ERROR: $x");
    exit (-1);
}

# -----
# dprint (list)
#
# prints if debugging is on
35 sub dprint
{
    40     if($gDebug)
    {
        print "(debug) ";
        print @_;
        print "\n";
    }
    45 }

sub dprintf
{
    50     if($gDebug)
    {
        print "(debug) ";
        printf @_;
    }
    55 }

sub date_time
{
    60     my ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdet) =
    localtime(time);
        $mon++;
        $year += 1900;

```

```

my $d = sprintf("%04d.%02d.%02d", $year, $mon, $mday);
my $t = sprintf("%02d:%02d:%02d", $hour, $min, $sec);

```

```

5     return "$d $t";
    }

```

```

10   sub basename
    {
        my $fileName = shift;
        my $baseName;

15       if($fileName =~ /\\[^\[\]*\]/)
        {
            $baseName = $1;
        }
20       else
        {
            $baseName = $fileName;
        }

25       return $baseName;
    }

```

```

30   # -----
    # print_command (string)
    #
    # prints time, and then string

```

```

35   sub print_command
    {
        my $command = shift;
        my $dt = date_time();

        print "# $dt (*) $command\n";
40     }

```

```

    sub print_warning
    {
        my $w1 = shift;

45     print "# mk_custom_sdk: WARNING";
        print " $w1\n";
    }

```

```

50   # -----
    # table_begin() returns a reference to a table to be constructed
    # table_addrow() takes the tableRef, and adds the rest of the
    #               args as elements to a single row
    # table_sprint() returns the table in left-justified columns
55   #               as one big string.
    #

```

```

    my $table_ROWBREAK = "##rzw##";          # an unlikely character sequence

```

```

60   sub table_begin
    {
        my @table;

```

T02T90"90T08860

```
return \@table;
}
```

```
5  sub table_addrow
    {
        my $tableRef = shift;
        my $eachArg;
        my $line;

10     while($eachArg = shift)
        {
            $line .= $table_ROWBREAK if $line;
            $line .= $eachArg;
        }

15     push @$tableRef,$line;
    }

sub table_sprint
20 {
    my $tableRef = shift;
    my $columnSpaces = 1;
    my @columnMaxWidths;
    my $line;
    my @line;
    my $i;
    my $len;
    my $result;

25     # first pass: find the maximum width for each line

    foreach $line (@$tableRef)
    {
        @line = split(/${table_ROWBREAK}/,$line);
        for($i = 0; $i < scalar(@line); $i++)
        {
            $len = length($line[$i]);
            $columnMaxWidths[$i] = $len if $columnMaxWidths[$i] < $len;
        }

30     }

    # second pass: spew them out with enough blanks

    foreach $line (@$tableRef)
    {
        @line = split(/${table_ROWBREAK}/,$line);
        for($i = 0; $i < scalar(@line); $i++)
        {
            my $lineItem = $line[$i];
            my $blanx = $columnMaxWidths[$i] - length($lineItem) + 1;

            $result .= $lineItem . (" " x $blanx);
        }
        $result .= "\n";

35     }

    return $result;
}

55

60 # -----
    # aNumber(hex-or-decimal)
    #
    sub aNumber
```

09880106 061201
T02120" 90108850

```

    {
    my $x = shift;

    return 1 if($x =~ /true/i or $x =~/yes/i);

5    $x = hex($x) if ($x =~ /^0+x/);
    $x = $x * 1.00;

    return $x;
10    }

# -----
# getAPTFNumber(ptfRef, path)
#
15 # convenience: extract child, convert to value
#
sub getAPTFNumber
    {
    my $ptfRef = shift;
20    my $path = shift;
    my $x;

    $x = get_data_by_path($ptfRef,$path);
    $x = aNumber($x);
25    return $x;
    }

30 sub usage
    {
        print <<EOP

EOP
35    }

# -----
# isModule(moduleRef)
#
40 # Given a ptf, is it an enabled MODULE?
# return 1 or 0.

sub isModule
45    {
        my $moduleRef = shift;

        return 0 if (get_name($moduleRef) ne "MODULE");
        return getAPTFNumber($moduleRef,"SYSTEM_BUILDER_INFO/Is_Enabled");
50    }

# -----
# readPTF(sourceFile)
#
# read in the PTF, and return a hash reference.
55 # Uses the module for all the real work.
#
# (return only the SYSTEM section)
#

60 sub readPTF
    {
        my $sourceFile = shift;
        my $ptfRef;
    }

```



```

print_warning("No bus master found. Pretending 32 bit wide bus.");
return 32;
}

```

```

5  # -----
   # getModuleKind(moduleRef)
   #
   # Return the module kind, like "uart"
   # or "ram".
10  # 2001Jan10: this comes from the MODULE/class
   # assignment, which presently reads juartwizard, &c.
   # This will be obsolete soon
   # 2001Jan31: We get the full name now, "altera_avalon_uart",
   # and elsewhere get the short struct name based on
15  # the name of the blah_struct.h file.
   # 2001Feb06: we have a global flag to get old names, for in-tree building

sub getModuleKind
{
20   my $moduleRef = shift;

   my $pKind;

   $pKind = get_data_by_path($moduleRef,"class");

25   if($gUseOldSillyJNames)
   {
       my %xlat =
       (
30         altera_avalon_uart => "juartwizard",
         altera_avalon_spi => "jspiwizard",
         altera_avalon_pio => "jpiowizard",
         altera_avalon_timer => "jtimerwizard",
         altera_avalon_i2x => "ji2xwizard",
35         altera_avalon_onchip_memory => "jramwizard",
         altera_avalon_user_defined_interface => "jusersocketwizard",
         altera_nios => "jnioswizard",
         altera_avalon_dev_board_flash => "jjam29lv800bwizard",
         altera_avalon_dev_board_flash_small =>
40         "jjam29lv800b_smallwizard",
         altera_avalon_sram16 => "j2xidt71v016s_smallwizard",
         altera_avalon_sram32 => "j2xidt71v016swizard",
         altera_avalon_nios => "jnioswizard",
         altera_avalon_nios => "jnioswizard",
45         altera_avalon_nios => "jnioswizard"
       );

       $pKind = $xlat{$pKind};
   }

   return $pKind;
50 }

# -----
# readAddressMap(ptfHashRef,germsAddressOut)
#
55 # Read in the source file, and return
# an associative array indexed by
# decimal base address; where each entry
# is a reference to an array, as follows:
#   [0] - peripheralKind
60 #   [1] - peripheralName
#   [2] - reference to hash of WIZARD_SCRIPT_ARGUMENTS
#   [3] - IRQ number if any
#   [4] - Lowest address

```

09830106 061201

09330105-051201
T02T90"

```
# [5] - just past Highest Address
# [6] - misc letter flags. 'E' means show the end address. 'D' decimal.
#

5 sub readAddressMap
{
  my $ptfRef = shift;
  my $germsAddressOut = shift;

10 my $key;
  my $pAddress;
  my $pIRQ;
  my $pEnd;
  my $pKind;
15 my @peripheralList;
  my $aPeripheralRef;

  my @printfUart;      # addresses for uart used by printf (and monitor)
  my @gdbUart;         # addresses for uart used by gdb
20 my @mainFlash;      # addresses for first flash memory found
  my $miscSysRef;

  my $masterDataWidth;

25 #
  # Establish address "na_null", which'll be a void *
  #
  $aPeripheralRef = [ "",
30     "na_null",
    "",
    "",
    0 ];
  push(@peripheralList, $aPeripheralRef);

35 $miscSysRef = getSystemSettings($ptfRef);

  # list the modules...
  {
40     my $moduleRef;
     my $childCount;
     my $i;

     $childCount = get_child_count($ptfRef);
45     for($i = 0; $i < $childCount; $i++)
     {
       $moduleRef = get_child($ptfRef, $i);
       #
       # Is it a MODULE?
50       #
       if(isModule($moduleRef))
       {
         my $pName;
         my $legalName;

55         $pName = get_data($moduleRef);
         $legalName = $pName;
         $legalName =~ tr/\-/_/;

         ($pAddress, $pEnd)
60         getModuleAddressRange($ptfRef, $moduleRef);

         # Found the germs?
```

09880106-061201
102790-9010880

```

    if (get_data_by_path($moduleRef, "WIZARD_SCRIPT_ARGUMENTS/Contents") eq
"germs")
5      {
        $$germsAddressOut = $pAddress if
$germsAddressOut;
      }

10    if (getAPTFNumber($moduleRef, "SYSTEM_BUILDER_INFO/Has_IRQ"))
      {
        $pIRQ =
getAPTFNumber($moduleRef, "SYSTEM_BUILDER_INFO/IRQ_Number");
      }
15    else
      {
        $pIRQ = 0;
      }
    $pKind = getModuleKind($moduleRef);

20    #
    # Assign this peripheral ref as a 5-element array
    reference []
    #

25    $aPeripheralRef = [$pKind,
                        "na_{$legalName}",

    get_child_by_path($moduleRef, "WIZARD_SCRIPT_ARGUMENTS"),
30    $pIRQ,
    $pAddress,
    $pEnd];

    #
    # Look for some various "special" peripherals...
    # If it's a UART, see if it qualifies as the printf
35    uart

    #
    if ($pName eq $$miscSysRef(maincomm_module))
      {
        @printfUart = @$aPeripheralRef;
      }
40    if ($pName eq $$miscSysRef(gdbcomm_module))
      {
        @gdbUart = @$aPeripheralRef;
      }
45    if ($pKind =~ /dev_board_flash/)
      {
        dprint "found a flash!";
        #
        # We have found a flash, but only by noting that
        # its name contains dev_board_flash. The only two
        # flash memories in the Nios 1.1 kit certainly
        # have that name. But it would be more general
        # to have a peripheral-property somewhere that
        # said, "Hey, I am a flash, and this is the name
        # of my write routine, and this is the name
        # of my erase routine. Yeah. dvb2001jan
        #
        @mainFlash = @$aPeripheralRef;
60    }

```

#

```

# Add it to the peripheral list
push(@peripheralList,$aPeripheralRef);
}
5      }
    }

if($sprintfUart[0] ne "")      # check by kind...
{
10      $aPeripheralRef = [$sprintfUart[0],
        "nasys_printf_uart",
        $sprintfUart[2],
        $sprintfUart[3],
        $sprintfUart[4],
15      $sprintfUart[5] ];
    push(@peripheralList,$aPeripheralRef);
}

if($gdbUart[0] ne "")      # check by kind...
20      {
        $aPeripheralRef = [$gdbUart[0],
        "nasys_gdb_uart",
        $gdbUart[2],
        $gdbUart[3],
        $gdbUart[4],
25      $gdbUart[5] ];
    push(@peripheralList,$aPeripheralRef);
}

if($mainFlash[0] ne "")
30      {
        $aPeripheralRef = [$mainFlash[0],
        "nasys_main_flash",
        $mainFlash[2],
        $mainFlash[3],
        $mainFlash[4],
35      $mainFlash[5],
        "E" ];
    push(@peripheralList,$aPeripheralRef);
40      }

    {
    #
    # Decide what range of RAM to use for programs & stack,
    # avoiding the vector table.
    #
    my $vecbase;
    my $vectop;
    my $programmibase;      # Could be ROM or RAM for compiled programs
50      my $programmientop;
    my $datamibase;      # Must be RAM for stack & variables (to be
    useful)
    my $datamientop;

    $vecbase = $$miscSysRef{vecbase};
    $vectop = $vecbase + 64 * (getMasterDataWidth($ptfRef) / 8);

    $programmibase = $$miscSysRef{programmibase};
    $programmientop = $$miscSysRef{programmientop};

60      $datamibase = $$miscSysRef{datamibase};
    $datamientop = $$miscSysRef{datamientop};

```

09830106-061201

in either.
More room above or below the vector table?
Do this for both program and data memory; vector table could be

```
#
5  if($vecbase - $datamembase > $datamemtop - $vectop)
    {
        $datamemtop = nb_min($datamemtop,$vecbase);
    }
else
10  {
        $datamembase = nb_max($datamembase,$vectop);
    }

    if($vecbase - $programmabase > $programmametop - $vectop)
15  {
        $programmametop = nb_min($programmametop,$vecbase);
    }
else
20  {
        $programmabase = nb_max($programmabase,$vectop);
    }

    $aPeripheralRef = ["",
25         "nasys_program_mem",
        "",
        0,
        $programmabase,
        $programmametop,
        "E" ];
30  push(@peripheralList,$aPeripheralRef);

    $aPeripheralRef = ["",
        "nasys_data_mem",
        "",
35         0,
        $datamembase,
        $datamemtop,
        "E" ];
    push(@peripheralList,$aPeripheralRef);
40

    $aPeripheralRef = ["",
        "nasys_stack_top",
        "",
45         0,
        $datamemtop];
    push(@peripheralList,$aPeripheralRef);

    $aPeripheralRef = ["",
50         "nasys_vector_table",
        "",
        0,
        $vecbase,
        $vectop,
        "E"];
55  push(@peripheralList,$aPeripheralRef);

    $aPeripheralRef = ["",
        "nasys_reset_address",
        "",
60         0,
        aNumber($$miscSysRef{reset_a})];
    push(@peripheralList,$aPeripheralRef);
```

09880106-061201

09380106-061201

```
5      $aPeripheralRef = ["",
        "nasys_clock_freq",
        "",
        0,
        aNumber($$miscSysRef{clock_freq}),
        "",
        "D"];
    push(@peripheralList, $aPeripheralRef);

10     $aPeripheralRef = ["",
        "nasys_clock_freq_1000",
        "",
        0,
        int(aNumber($$miscSysRef{clock_freq}) / 1000),
        "",
        "D"];
15     push(@peripheralList, $aPeripheralRef);
    }

20     return @peripheralList;
    }

25     # -----
    # mkSDKTree(directory for new tree)
    #
    # Build the empty directory structure
    # Any files already there, just leave.

30     sub rmdir_if_possible    # return 0 for success, > 0 for missed files
    {
        my $dir = shift;
        my @fileList;
        my $file;
35         my $misses = 0;          # how many we failed to kill

        if(-e $dir)
        {
40             opendir (DIR,$dir) or $misses++;
            @fileList = readdir(DIR);
            closedir (DIR);

            foreach $file (@fileList)
            {
45                 next if ($file eq "." or $file eq "..");

                 if( -d "${dir}/${file}")
                 {
50                     $misses += rmdir_if_possible("${dir}/${file}");
                 }
                 else
                 {
55                     unlink("${dir}/${file}") or $misses++;
                 }
            }
            rmdir($dir) or $misses++;
        }

60         return $misses;
    }

    sub mkdir_if_needed
```

```

    {
    my $dir = shift;
    my $mode = shift;

5      if(! -e $dir)
        {
            mkdir($dir,$mode) or return;
        }
    return "ok";
10    }

sub mkSDKTree
    {
    my $dir = shift;
15    my $misses;

    #
    # The "src" directory belongs to the end user,
    # he or she may put any code they like there,
20    # and expect it to persist across builds.
    #
    # However, the lib, inc, and doc, are
    # ours, all ours, and we can't have deleted
    # old peripherals malingering in there,
    # so we delete them.
25    #

    mkdir_if_needed($dir,511) or return;

30    $misses = rmdir_if_possible("${dir}/lib");
    print_warning "Could not delete lib; $misses left" if($misses);

    $misses = rmdir_if_possible("${dir}/inc");
    print_warning "Could not delete inc; $misses left" if($misses);
35    $misses = rmdir_if_possible("${dir}/doc");
    print_warning "Could not delete doc; $misses left" if($misses);

    mkdir_if_needed($dir."/inc",511) or return;
    mkdir_if_needed($dir."/lib",511) or return;
40    mkdir_if_needed($dir."/doc",511) or return;
    mkdir_if_needed($dir."/src",511) or return;

    return "ok";
45    }

# -----
50 # findStructFile(components_directories,dirName,fileTail)
#
# Given a directory path, return the name of a file,
# if any, which ends with fileTail
#
55 #
sub findStructFile
    {
    my $components_directories = shift;
    my $dirName = shift;
60    my $fileTail = shift;

    my @components_directories;
    my $components_directory;

```



```

my @dirList;
my $fileName;
my $shortKindName;

5    @components_directories = split(/\/+/, $components_directories);

    foreach $components_directory (@components_directories)
    {
dprint "findStructFile: searching $components_directory";
10    if(opendir DIR, "${components_directory}/${dirName}")
        {
            @dirList = readdir DIR;
            closedir DIR;

15            foreach $fileName (@dirList)
                {
dprint "findStructFile: searching $components_directory 2";
                    if($fileName =~ /^(.*)_${fileTail}$/)
                        {
20                            $shortKindName = $1;
dprint "findStructFile: found $shortKindName structure";
                            return
                                ("${components_directory}/${dirName}/${fileName}", $shortKindName);
                        }
                    }
                }
            }
        }
    }
    return;
}

30 # -----
#
generateAddressMap(components_directories, fileName, @$peripheralListRef, ptfName,
ptfRef)
#
35 sub generateAddressMap
    {
        my $components_directories = shift;
        my $fileName = shift;
        my $peripheralListRef = shift;
        my $ptfName = shift;
        my $ptfRef = shift;
        my $pAddress;
45        my $pEnd;
        my $pIRQ;
        my $pName;
        my $pKind;
        my $pFlags;
50        my $dt = date_time();
        my $i;
        my $nios_system_name = get_data($ptfRef);
        my $monitor_string;

55        #
        # Painfully retrieve just the monitor id string
        #
        {
            my $miscSysRef;

60            $miscSysRef = getSystemSettings($ptfRef);
            $monitor_string = $$miscSysRef{germs_monitor_id};
            $monitor_string = $nios_system_name if !$monitor_string;
        }
    }

```

}

my \$baseName = basename(\$fileName);

5 open HFILE,">\$fileName.h" or return;
 open SFILE,">\$fileName.s" or return;

 print HFILE <<EOP;

10 /*
 * File: \$baseName.h
 *
 * This file is a machine generated address map
 * for a Nios hardware design.
 * \$ptfName
15 *
 * Generated: \$dt
 */

20 #ifndef _\${baseName}_
 #define _\${baseName}_

 #ifdef __cplusplus
 extern "C" {
 #endif

25 EOP

 print SFILE <<EOP;

30 ;
 ; File: \$baseName.s
 ;
 ; This file is a machine generated address map
 ; for a Nios hardware design.
 ; \$ptfName
35 ;
 ; Generated: \$dt
 ;

40 ; Simple macro to equate & globalize at once
 .macro GEQU sym,val
 .global \\sym
 .equ \\sym,\\val
 .endm

45 EOP

 my \$hTableRef = table_begin();
 my \$sTableRef = table_begin();

50 for(\$i = 0; \$i <= \$\$peripheralListRef; \$i++)
 {
 my \$p = \$\$peripheralListRef[\$i];
 my \$cType;
55 my \$isRAMLike;
 my \$structFile;
 my \$hasStructFile;
 my \$shortPKind;
 my \$pAddressString;

60 \$pKind = \$\$p[0];
 \$pName = \$\$p[1];
 \$pIRQ = \$\$p[3];

09330406.061201

```

$Address = $$p[4];
$End      = $$p[5];
$Flags    = $$p[6];

```

```

5      ($structFile,$shortPKind) = findStructFile (
                                $components_directories,
                                "${pKind}/${CUSTOM_SDK_PIECES_DIR}",
                                "struct.h");

```

```

10     # arbitrary: "RAM-like" spans are > some minimum
    $isRAMLike = ($End-$Address > 4095);
    $isRAMLike |= $Flags =~ /E/;
    $hasStructFile = (-e $structFile);

```

```

15     if($Flags =~ /D/)
    {

```

```

        $cType = "long";
    }

```

```

    elseif($pKind eq "" or $pKind =~ / / or $isRAMLike or

```

```

20 ! $hasStructFile)
    {

```

```

        $cType = "void *";
    }

```

```

    else
    {

```

```

        $cType = "np_${shortPKind} *";
    }

```

```

    $AddressString = ($Flags =~ /D/) ?
        sprintf("%d", $Address)
        : sprintf("0x%08x", $Address);

```

```

    table_addrow($hTableRef,
        "#define",
        "$pName",
        "((($cType))",
        "${AddressString}))";

```

```

    table_addrow($sTableRef,
        "    GEQU",
        "$pName",
        ",",
        $AddressString,
        "; $pKind");

```

```

45     if($isRAMLike)    # only report ends of "RAM-like" spans
    {

```

```

        table_addrow($hTableRef,
            "#define",
            "${pName}_end",
            "((($cType))",
            (sprintf("0x%08x", $End)) );

```

```

        table_addrow($sTableRef,
            "    GEQU",
            "${pName}_end",
            ",",
            (sprintf("0x%08x", $End)) );
    }

```

```

60     if($pIRQ)
    {
        table_addrow($hTableRef,

```

09330105 "051201

09880105-061201
TOTAL: 907860

```

        "#define",
        "${pName}_irq",
        " ",
        $pIRQ);

5          table_addrow($sTableRef,
                "      GEQU",
                "${pName}_irq",
                " ",
10          $pIRQ);
        }
    }

15    {
        my $x;

        $x = table_sprint($hTableRef);
        print HFILE $x;

20        $x = table_sprint($sTableRef);
        print SFILE $x;
    }

    #
    # Print a macro for getting the system name
    #
25    print SFILE <<EOP;

    .macro nm_system_name_string
    .asciz "$nios_system_name"
30    .endm

    .macro nm_monitor_string
    .asciz "$monitor_string"
35    .endm

EOP

    print HFILE <<EOP;

40    // Parameters for each peripheral.

EOP

    print SFILE <<EOP;

45

; Parameters for each peripheral.

EOP

50    for($i = 0; $i < $$peripheralListRef; $i++)
    {
        my $p = $$peripheralListRef[$i];

55        $pKind = $$p[0];
        $pName = $$p[1];

        if($pKind and $pName =~ /^na_/ and $$p[2])
        {

60            printf HFILE "\n// Parameters for %s named %s at 0x%08x:\n",
                $pKind,
                $pName,
```

\$pAddress;

printf SFILE "\n; Parameters for %s named %s at 0x%08x:\n",
\$pKind,
\$pName,
\$pAddress;

printColumns(*HFILE,\$\$p[2],"// "," " = ","");
printColumns(*SFILE,\$\$p[2]," "; "," = ","");
}

print HFILE <<EOP;

#ifdef __cplusplus
}
#endif

#endif // _\${baseName}_

/* end of file */
EOP

print SFILE <<EOP;

; end of file
EOP

close HFILE;
close SFILE;

return "ok";
}

generatePeripheralsStructs(basename,
components_directories, systemName)
#

\@peripheralList,

sub generatePeripheralsStructs

{
my \$fileName = shift;
my \$peripheralListRef = shift;
my \$components_directories = shift;
my \$ptfName = shift;

my \$dt = date_time();
my \$pAddress;
my \$pName;
my \$pKind;
my \$lib_files_found;
my \$structureContents;
my \$baseName;
my \$i;
my %peripheralKinds;

\$baseName = basename(\$fileName);
open HFILE,">\$fileName.h" or return;

09380106 "061201

binmode HFILE;

open SFILE,">\$fileName.s" or return;
binmode SFILE;

print HFILE <<EOP;

/*
* File: \$baseName.h
*
* This file is a machine peripherals definition
* for a Nios hardware design.
* \$ptfName
*
* Generated: \$dt
*/

#ifndef _\${baseName}_
#define _\${baseName}_

#ifdef __cplusplus
extern "C" {
#endif

EOP

print SFILE <<EOP;

;
; File: \$baseName.s
;
; This file is a machine peripherals definition
; for a Nios hardware design.
; \$ptfName
;
; Generated: \$dt
;

EOP

Figure out which ones are actually present
#

for(\$i = 0; \$i < \$\$peripheralListRef; \$i++)
{
my \$p = \$\$peripheralListRef[\$i];

\$pKind = \$\$p[0];
\$pName = \$\$p[1];

\$peripheralKinds{\$pKind} ++ if (\$pName =~ /^na_/); # only count
na_, not nasys's
}

Find the <x>_struct.h file for each of them, now.

foreach \$pKind (sort(keys(%peripheralKinds)))
{

my \$structFile;

(\$structFile) = findStructFile (
\$components_directories,

```
"${pKind}/${CUSTOM_SDK_PIECES_DIR}",
"struct.h");
```

```
$structureContents =
    readFile($structFile);
```

```
print HFILE "// $pKind: $peripheralKinds($pKind) present \n";
print HFILE $structureContents, "\n";
```

```
($structFile) = findStructFile (
    $components_directories,
    "${pKind}/${CUSTOM_SDK_PIECES_DIR}",
    "struct.s");
```

```
$structureContents =
    readFile($structFile);
```

```
print SFILE "; $pKind: $peripheralKinds($pKind) present \n";
print SFILE $structureContents, "\n";
}
```

```
print HFILE <<EOP;
```

```
#ifdef __cplusplus
}
```

```
#endif
```

```
#endif // _${baseName}_
```

```
/* end of file */
```

```
EOP
```

```
close HFILE;
```

```
print SFILE <<EOP;
```

```
; end of file
```

```
EOP
```

```
close SFILE;
```

```
return "ok";
```

```
}
```

```
# -----
```

```
# generateNiosHeader(headerFileName, ptfName)
```

```
#
```

```
# Generate nios.h and nios.s, which just include
# the other three header files.
```

```
#
```

```
sub generateNiosHeader
```

```
{
```

```
my $fileName = shift;
```

```
my $ptfName = shift;
```

```
my $dt = date_time();
```

```
my $baseName = basename($fileName);
```

```
open HFILE, ">$fileName.h" or return;
binmode HFILE;
```

```
open SFILE, ">$fileName.s" or return;
binmode SFILE;
```

09380106-061201

09880105 051201
T02T90" 90T0880

```

        print HFILE <<EOP;
/*
 * File: $baseName.h
 *
5  * This file is a machine generated header
 * for a Nios hardware design.
 * $ptfName
 *
10  * Generated: $dt
 */

#ifndef _${baseName}_
#define _${baseName}_

15  #include "nios_map.h"
#include "nios_peripherals.h"

#endif // _${baseName}_

20  /* end of file */
EOP
        close HFILE;

        print SFILE <<EOP;
25  ;
; File: $baseName.s
;
; This file is a machine generated header
; for a Nios hardware design.
30  ; $ptfName
;
; Generated: $dt
;

        .include "nios_macros.s"
35  .include "nios_peripherals.s"
        .include "nios_map.s"

; end of file
EOP
40  close SFILE;
}

# -----
45  # findDirInDirs(components_directories,pKind)
#
# given the +-separated components directories, return
# a path to <something>/pKind, if one exists. Else "".
#
50  sub findDirInDirs
{
    my $components_directories = shift;
    my $pKind = shift;

55  my @components_directories;
my $components_directory;
my $p;

    @components_directories = split(/\+/, $components_directories);
60  foreach $components_directory (@components_directories)
    {
        $p = "${components_directory}/${pKind}";
    }
}
```



```
return $p if( -e $p)
}
```

```
return "";
}
```

```
# -----
# addLibDocInc_and_SrcFiles(sdkDir, ptfReference, components_directories)
#
# From each wizard's "custom_sdk_pieces" subtree,
# extract out each and every lib, doc, inc, and src file.
# If the file already exists on the other side, don't replace it.
```

```
sub addLibDocInc_and_SrcFiles
```

```
{
    my $sdkDir = shift;
    my $ptfRef = shift;
    my $components_directories = shift;
```

```
    my $moduleRef;
    my $pKind;
```

```
    my $piecesDir;    # directory within j<x>wizard with files to copy over
    my $subDir;
    my $i;
    my $childCount;
```

```
    $childCount = get_child_count($ptfRef);
    for($i = 0; $i < $childCount; $i++)
```

```
    {
        $moduleRef = get_child($ptfRef,$i);
        next if (!isModule($moduleRef));
```

```
        $pKind = getModuleKind($moduleRef);
        $piecesDir = findDirInDirs($components_directories,$pKind);
        next if (!$piecesDir);
        $piecesDir .= "/${CUSTOM_SDK_PIECES_DIR}";
        foreach $subDir ("lib","doc","inc","src")
```

```
        {
            copyDirContents("${piecesDir}/${subDir}", "${sdkDir}/${subDir}");
        }
    }
}
```

```
return "ok";
}
```

```
# -----
# getModuleAddressRange(ptfRef,module)
```

```
#
# module can be either moduleRef or moduleName
#
```

```
sub getModuleAddressRange
```

```
{
    my $ptfRef = shift;
    my $moduleName = shift;
```

```
    my $infoRef;
    my $addrExponent;
    my $baseAddr;
    my $addrSpan;
    my $stopAddr;
    my $alignment;
```

```

my $wordSizePerAddress;
my $bytesPerAddress;
my $addressAlignment;

5      #
      # Discern moduleRef from moduleName
      #

      if(ref($moduleName) eq "HASH")
10      {
          $infoRef = get_child_by_path($moduleName, "SYSTEM_BUILDER_INFO");
          $moduleName = get_data($moduleName);
      }
      else
15      {
          $infoRef = get_child_by_path($ptfRef, "MODULE
$(moduleName)/SYSTEM_BUILDER_INFO");
      }

20      $baseAddr = getAPTFNumber($infoRef, "Base_Address");
      $addrSpan = getAPTFNumber($infoRef, "Address_Span");

      $addressAlignment = get_data_by_path($infoRef, "Address_Alignment");
      if($addressAlignment eq "dynamic")
25      {
          $wordSizePerAddress = getAPTFNumber($infoRef, "Data_Width");
      }
      elsif($addressAlignment eq "native")
30      {
          $wordSizePerAddress = getMasterDataWidth($ptfRef);
      }
      elsif($addressAlignment eq "word")
35      {
          $wordSizePerAddress = 32;
      }
      elsif($addressAlignment eq "halfword")
40      {
          $wordSizePerAddress = 16;
      }
      elsif($addressAlignment eq "byte")
45      {
          $wordSizePerAddress = 8;
      }
      else
50      {
          # default to width of master (Nios) CPU
          # print_warning("$moduleName
Address_Alignment=\ "$addressAlignment\";");
          $wordSizePerAddress = getMasterDataWidth($ptfRef);
55      }

      $bytesPerAddress = int($wordSizePerAddress / 8);
      $addrExponent = getAPTFNumber($infoRef, "Address_Width");

      $stopAddr = $baseAddr + (1 << $addrExponent) * $bytesPerAddress;
      $stopAddr = $baseAddr + $addrSpan if $addrSpan;

      return ($baseAddr, $stopAddr);
60      }

      # -----
      # getSystemSettings(ptfRef)
      #

```

09880106.0612001

```

# return a reference to the associative
# array which defines the processor options,
# including reset address, vecbase, 16/32, and mstep.
#
5  # Return reference to array of various
# useful globalish information about the system module
#
# FIXME: this only works if the bus master is a Nios.
#
10 sub getSystemSettings
    {
        my $ptfRef = shift;
        my $moduleRef;
        my $pKind;
15     my %result;
        my $i;
        my $childCount;

        my $moduleName;
        my $offset;
        my $base;
        my $top;

        # First, find the processor within the ptfRef
        $childCount = get_child_count($ptfRef);
        for($i = 0; $i < $childCount; $i++)
        {
            $moduleRef = get_child($ptfRef,$i);
            next if (!isModule($moduleRef));
30
            $pKind = getModuleKind($moduleRef);
            if($pKind eq "altera_nios" or $pKind eq "jnioswizard")
            {
                my $x;
                # When found, fill the result
35 dprint "found altera_nios";

                # Vector Table

                $moduleName
40 get_data_by_path($moduleRef, "WIZARD_SCRIPT_ARGUMENTS/vecbase_module");
                $offset
                getAPTFNumber($moduleRef, "WIZARD_SCRIPT_ARGUMENTS/vecbase_offset");
                ($base,$top) = getModuleAddressRange($ptfRef,$moduleName);
                $result{vecbase} = $base + $offset;
45 dprint "module = $moduleName, vecbase = $base, vectop = $top";

                # Reset Address

                $moduleName
50 get_data_by_path($moduleRef, "WIZARD_SCRIPT_ARGUMENTS/reset_module");
                $offset
                getAPTFNumber($moduleRef, "WIZARD_SCRIPT_ARGUMENTS/reset_offset");
                ($base,$top) = getModuleAddressRange($ptfRef,$moduleName);
55 $result{reset_a} = $base + $offset;

                # MSTEP and MUL

                $x
60 getAPTFNumber($moduleRef, "WIZARD_SCRIPT_ARGUMENTS/mstep");
                $x = $x ? 1 : 0;
                $result{mstep} = $x;          # 1 or 0
    }

```

==

5

11

10

15

20

25

30

11

35

11

40

45

50

55

60

```

my $germsAddress = shift;

my $libDir = "${sdkDir}/lib";
my @libFiles;
my $libName;
my $file;
my $key;
my $dt = date_time();

my $nios_system_name = get_data($ptfRef);

my $nios_data_bits;      # 16 or 32 bit processor
my $nios_mstep_support;
my $nios_multiply_support;

my %makefileVars; # extra values set in the makefile, and passed down to
programs.

#
# Set germsAddress to a number in hex
#
$germsAddress = aNumber($germsAddress);
$germsAddress = sprintf("0x%08x", $germsAddress);

#
# First, find the processor within the ptfRef
#
{
my $miscSysRef;

$miscSysRef = getSystemSettings($ptfRef);
$nios_data_bits = $$miscSysRef{bits};
$nios_mstep_support = $$miscSysRef{mstep};
$nios_multiply_support = $$miscSysRef{multiply};
}

$libName = "libnios\$(M).a";

$makefileVars{NIOS_USE_MSTEP} = $nios_mstep_support . " # CPU option
(shift, test, & add)";
$makefileVars{NIOS_USE_MULTIPLY} = $nios_multiply_support . " # CPU
option (16x16->32)";
$makefileVars{NIOS_USE_FAST_MUL} = 1 . " # Faster but larger int
multiply routine";
$makefileVars{NIOS_USE_SMALL_PRINTF} = 1 . " # Smaller non-ANSI printf,
with no floating point formats";
$makefileVars{NIOS_USE_CWPMGR} = 1 . " # Turn off to disable underflow
handling (dangerous)";
$makefileVars{NIOS_USE_CONSTRUCTORS} = 1 . " # Call c++ static
constructors; turn off for smaller footprint";

$makefileVars{NIOS_SYSTEM_NAME} = $nios_system_name;
$makefileVars{NIOS_MONITOR} = "nios_germs_monitor";

#
# spew a makefile, most uglywise
#
opendir (DIR,$libDir) or return;
@libFiles = readdir (DIR);
closedir (DIR);

open (FILE,">${libDir}/Makefile") or bail;;

```

09380106 061201

```

        print FILE <<EOP;

5   #
    # Nios SDK Generated Makefile
    # $dt
    # $ptfName
    #
    EOP
10   foreach $key (sort(keys(%makefileVars)))
        {
            print FILE "${key} = %makefileVars{$key}\n";
        }
        print FILE <<EOP;

15   AS = nios-elf-as
    CC = nios-elf-gcc -c -O2 -g
    AR = nios-elf-ar
    M = ${nios_data_bits}
20   OBJ = ./obj/${M}
    SRC = .
    E = echo ----

    EOP

25   {
        my $aF;
        my $cF;

30   $aF = "--defsym __nios\${M}__=1 -m\${M} -I ../inc";
        $cF = " -m\${M} -I ../inc";
        foreach $key (sort(keys(%makefileVars)))
            {
35   $aF .= " \\n\t--defsym __" . lc($key) . "__=\${$key}";
            $cF .= " \\n\t-D __" . lc($key) . "__=\${$key}";
            }

        print FILE "ASFlags = ${aF}\n\n";
        print FILE "CCFlags = ${cF}\n\n";
40   }

        print FILE <<EOP;
    RESULT = $libName

45   OBJECTS = \
    EOP

        foreach $file (@libFiles)
            {
50   my $basename;

            if($file =~ /^(.*)\[cs\]$/
                and $file ne "nios_germes_monitor.s")
            {
55   $basename = $1;
                print FILE "\t\${OBJ}/${basename}.o \\n";
            }
        }
        print FILE "\n\n";

60   print FILE <<EOP;

    \${OBJ}/%.o : \${SRC}/%.s

```

09830106-051201

```

    \$(E) Assembling \$<
    \$(AS) \$(ASFlags) \$< -o \$@
    \$(E) Archiving \$@
    \$(AR) -r \$(RESULT) \$@
5
\$(OBJ)/%.o : \$(SRC)/%.c
    \$(E) Compiling \$<
    \$(CC) \$(CCFlags) \$< -o \$@
    \$(E) Archiving \$@
10    \$(AR) -r \$(RESULT) \$@

\$(OBJ) :
    \$(E) Making \$@ Directory
    mkdir \$@
15

clean : \$(OBJ)
    \$(E) Removing objects
    rm -f ../../\$(NIOS_SYSTEM_NAME)_germs_monitor.mif
    rm -f \$(OBJECTS) \$(RESULT)
20

\$(RESULT) : \$(OBJ) \$(OBJECTS)
    \$(E) Build \$@

GERMSMON :
25    \$(E) Building Germs monitor \$(NIOS_MONITOR)\@{\germsAddress}
    nios-build -s -b \$germsAddress \$(NIOS_MONITOR).s -o
    \$(OBJ)/\$(NIOS_MONITOR)
    rm -rf \$(NIOS_MONITOR).s.o
    \$(E) Converting to \$(NIOS_SYSTEM_NAME)_germs_monitor.mif
30    nios-convert \$(OBJ)/\$(NIOS_MONITOR).srec --oformat=mif --width=\$(M)
    mv \$(OBJ)/\$(NIOS_MONITOR).mif
    ../../\$(NIOS_SYSTEM_NAME)_germs_monitor.mif

all : clean \$(RESULT) GERMSMON
35

# end of file
EOP

    close (FILE);
40

}

45 # -----
# printColumns(FILERef,ptfRef,lineBeginString,columnbreakString,lineEndString)
#

sub printColumns
50 {
    my \$FILE = shift;
    my \$ptfRef = shift;

    my \$lineBeginString = shift;
55    my \$columnBreakString = shift;
    my \$lineEndString = shift;

    my \$nameWidth = 0;
    my \$dataWidth = 0;
60    my \$name;
    my \$data;

    my \$i;
```

```

my $childCount;
my $childRef;

$childCount = get_child_count($ptfRef);
for($i = 0; $i < $childCount; $i++)
{
    my $w = 0;

    $childRef = get_child($ptfRef,$i);
    $name = get_name($childRef);
    $data = get_data($childRef);

    $w = length($name);
    $nameWidth = $w if $w > $nameWidth;

    $w = length($data);
    $dataWidth = $w if $w > $dataWidth;
}

for($i = 0; $i < $childCount; $i++)
{
    my $w = 0;

    $childRef = get_child($ptfRef,$i);
    $name = get_name($childRef);
    $data = get_data($childRef);

    print $FILE $lineBeginString;
    print $FILE " " x ($nameWidth - length($name));
    print $FILE $name;
    print $FILE $columnBreakString;
    print $FILE $data;
    print $FILE $lineEndString;
    print $FILE "\n";
}
}

```

```

# -----
45 # parseArgs
#
# Given a list of arguments, return
# a hash where the keys and values
# are taken from those arguments of
50 # the form "--key=value". The hyphens
# disappear from the key name.
#
# A command line switch of "--key"
# is equivalent to "--key=1".
55 #
# a special key named _argc contains
# a count of non-dash-dash arguments,
# and they are in the hash as {0}, {1},
# and so on.
60 sub parseArgs
{
    my $arg;

```

09880106 "061201


```

my $argVal;
my $argc;
my %hash;

```

```

5      $argc = 0;

```

```

while($arg = shift)
{
10      usage if $arg eq "--help";

      if($arg =~ /^--/)
      {
15          if($arg =~ /^--(.*)\=(.*)$/)
          {
              $arg = $1;
              $argVal = $2;
          }
          else
20          {
              $argVal = 1;
          }

          $hash{$arg} = $argVal;
25      }
      else
      {
          $hash{$argc++} = $arg;
30      }

      $hash{_argc} = $argc;

      return %hash;
35  }

```

```

# -----
# getSwitch(hashRef, switchName, defaultValue [, mustBeNumber])
#
40 # Look at a hash as returned by parseArgs, and
# give the value of the switch, or the defaultValue
# if it was not specified in the command line.

```

```

sub getSwitch
45  {
      my $hashRef = shift;
      my $switchName = shift;
      my $defaultValue = shift;
      my $mustBeNumber = shift;

50      my $switchValue;

      $switchValue = $$hashRef{$switchName};
      $switchValue = $defaultValue if ($switchValue eq "");
55      $switchValue *= 1 if ($mustBeNumber);

      return $switchValue;
  }

```

```

60

```

```

# -----
#
# Necessary inputs:

```

09830106-061201

```

# --sopc_directory=dir      complete path to directory containing jnioswizard
& siblings
# --system_name=name        name of the system; add .ptf for the file we need
# --system_directory=dir    path to the ptf file
5 #
# Optional inputs:
# --sdk_directory=dir       complete path to custom sdk result directory. All
enclosing
#                             directories above it must already exist.
10 #                             default: system_directory/system_name_sdk
# --sopc_lib_dir=dir[(+dir)*]
#                             list of directories for components
# --nios_sh=full path       what nios_sh to source for paths, &c.
#                             default: /usr/altera/excalibur/nios-sdk/nios_sh
15 # --debug=1 (or 0)        turn on debug print messages. defaults to off.

sub mk_custom_sdk
{
20   my $sopc_directory;
   my $system_name;
   my $system_directory;
   my $sdk_directory;
   my $nios_sh;
   my $altera_dir;
25   my $ptf_name;
   my $result = 0;

   my $components_directories;

30   my %switches;

   %switches = parseArgs(@_);

   $sopc_directory = getSwitch(\%switches,"sopc_directory",".");
35   $system_name = getSwitch(\%switches,"system_name","a_nios_system");
   $system_directory = getSwitch(\%switches,"system_directory",".");
   $sdk_directory = getSwitch(\%switches,
                               "sdk_directory",
                               "${system_directory}/${system_name}_sdk");
40   $components_directories = getSwitch(\%switches,
                                         "sopc_lib_path",
                                         "${sopc_directory}/components");
   $altera_dir = getSwitch(\%switches,
                           "altera_dir",
45   "");
   $nios_sh = getSwitch(\%switches,
                        "nios_sh",
                        "");

50   $gUseOldSillyJNames = getSwitch(\%switches,"use_old_wizard_names",0);

   $gDebug = getSwitch(\%switches,"debug",$gDebug);

   $ptf_name = "${system_directory}/${system_name}.ptf";

55   my $germsAddress;
   my $ptfRef;      # the whole PTF, hashed & such
   my @peripheralList; # indexed by base address, in decimal,
   "type:instanceName"

60   if($sdk_directory eq "")
   {
       print_command "not enough arguments.";
   }
}

```

09880106 "061201

```
    return 0;
}
```

```
5    # Read project description for memory map and name.
    # Discern which peripheral classes are needed.

    print_command "Reading project ${ptf_name}.";

10    $ptfRef = readPTF($ptf_name);

    @peripheralList = readAddressMap($ptfRef, \ $germsAddress);

    if (scalar(@peripheralList) <= 0)
15        {
            bail "No peripherals.";
        }

    # Generate Customized SDK directory structure

20    print_command "Generating SDK Tree";
    mkSDKTree($sdk_directory) or bail "Could not create SDK tree.";

    # log beginning
    open (FILE, ">>${sdk_directory}/custom_sdk_log.txt") or return;
    print FILE date_time, ": Started latest custom_sdk\n";
    close FILE;

    # Create nios_map.h

30    print_command "Generating address map header file.";
    generateAddressMap($components_directories, "${sdk_directory}/inc/nios_map
", \@peripheralList, $ptf_name, $ptfRef)
        or bail "Could not generate address map.";

35    # Create nios_peripherals.h

    print_command "Generating peripherals structures file.";
    generatePeripheralsStructs("${sdk_directory}/inc/nios_peripherals",
40        \@peripheralList, $components_directories, $ptf_name)
        or bail "Could not generate peripherals structures file.";

    # Generate nios.h and nios.s, which just combines them

45    generateNiosHeader("${sdk_directory}/inc/nios", $ptf_name);

    # Copy files into lib, doc, inc, and src.

50    print_command "Adding lib, doc, inc, and src files.";
    addLibDocInc_and_SrcFiles($sdk_directory, $ptfRef, $components_directories)
;

    # Generate a makefile in the lib directory

55    print_command "Generating Makefile in lib.";
    generateMakefileInLib($sdk_directory, $ptfRef, $ptf_name, $germsAddress);

    # log end
    open (FILE, ">>${sdk_directory}/custom_sdk_log.txt") or return;
    print FILE date_time, ": Finished latest custom_sdk\n";
    close FILE;

60
```

09080106.061201

```

#
# Build the darned thing
#
5      {
      my $bin_directory = "${sopc_directory}/bin";
      my $ssh;

      #
      # We only allow Cygwin to be installed
10     # in c:\cygwin or d:\cygwin, so look
      # in those allowed places.
      #
      # After that, try unixy locations.
      #

15     if($^O eq "MSWin32")
        {
          $ssh = 'c:\cygwin\bin\sh.exe';
          $ssh = 'd:\cygwin\bin\sh.exe' if (! -e $ssh);
20         }
      else
        {
          $ssh = '/bin/sh';
        }

25     if(! -e $ssh)
        {
          print_warning "You must install Cygwin & Nios SDK 1.1 to get
a library and Germs monitor.";
30         $result = -1;
        }
      else
        {
          print_command "Making Library & Germs Monitor";
35         my $system_command;

          $system_command = "${sh} -c \"cd $sdk_directory/lib\";

          $system_command .= ";nios_sh=${nios_sh}" if $nios_sh ne "";
40         $system_command .= ";altera=${altera_dir}" if $altera_dir ne
          "";

          $system_command .= ";. ${bin_directory}/mk_custom_sdk.sh\"";
45         #
          # Tell modified perl to spawn new process invisibly
          #

          open (ABRAHAM_LINCOLN_STEALTH, "");
          close ABRAHAM_LINCOLN_STEALTH;

          dprint "system_command is $system_command";
          $result = system ($system_command);
55         #
          # Turn off invisible-flag
          #

          open (ABRAHAM_LINCOLN_NO_STEALTH, "");
          close ABRAHAM_LINCOLN_NO_STEALTH;
60         if($result)

```

09080106 "061201

```

{
    print_warning "Make failed (Nios SDK required... did
you install it?)";
    $result = -1;
}
}

return $result;
}

# -----
# The Body
#

return "ok";

# end of file
```

09880106.061201
T02T90" 90T08850

Mk_Nios.pm

```
use wiz_utils;
use wiz_convert;
```

```
5 #####
#
# Mk_Nios
#
10 # Builds a Nios core from named arguments, including all
# design and support files, plus synthesis script.
#
$Mk_Nios_Doc=<<END_OF_DOCUMENTATION_STRING ;
# LONG NAME      SHORT NAME      DEFAULT      DESCRIPTION
15 # -----
# *name          nm              --none--      Name
# project_dir    proj           .             Project directory for output files.
# bits           bits          32            *(32|16)* Number of ALU bits.
# num_regs       regs          256          *(128|256|512)* Reg file size.
20 # shift_size    shift         7            *(1|3|7|15|31)* Bits/clock shift speed
# mstep          mstep         YES          *(YES|NO)* Include MSTEP unit?
# multiply        multiply      NO           *(YES|NO)* Include multiply unit?
# wvalid_wr      wr_wv         NO           *(YES|NO)* Writeable WVALID reg?
# *high_a        ha            --none--      Highest mem address in system.
# *vecbase       vb            --none--      Vector table base address.
25 # *reset_a      ra            --none--      Reset execution-start address.
# idle_cycle     idle_cycle    YES          *(YES|NO)* Insert Rd->wr idle cycle?
# mm_span        mm_span       --none--      main memory address span.
# mm_base        mm_base       --none--      main memory base address.
30 # leo_settings  leo_set       $DEFAULT_LEO_SETTINGS Synthesis-settings script.
# user_instruction_list uil     --none--      delimited list of user instr.
# part_type      --none--      apex20e      part type used for firm flip flops

END_OF_DOCUMENTATION_STRING
35 # This was used for testing:
# user_instruction_list uil RR-K-011100-FMUL makes FMUL
#
# But the first argument to this function (NOT a named argument)
# tells us whether to build a Nios core, HDL files and all,
40 # or to build only a PTF-file.
#
# The PTF_ONLY parameter must be "YES" or "NO."
#
#
45 #####
sub Mk_Nios
{
    my ($PTF_ONLY, @named_arg_list) = (@_);

50     my ($named_arg_string, $arg, $user_defined) =
        Process_Wizard_Script_Arguments ($Mk_Nios_Doc, @named_arg_list);

    my $part_type = $$arg{part_type};
    $part_type =~ tr/A-Z/a-z/;
55     # We create several Firm_Flip_Flop variants so that we can, for example,
    # assign Fast I/O register attributes to them later.
    #
    # These global variables we're setting get used
    # inside the nios-core Vpp script. David Van Brink would be horrified.
60     #
    # These names used to be long and luxurious. Now they're short
    # and spartan. FPGA Express gets angry if your name runs over 32
    # characters, and the old long names "used up" 16 precious characters.
```

09830106-061201

Now they only use-up three. But they're short. And spartan. Sorry.

```
#
$ADDRESS_OUT_REG_MODULE_NAME = $$arg{name} . "_ar"; # _address_out_reg
$CONTROL_OUT_REG_MODULE_NAME = $$arg{name} . "_cr"; # _control_out_reg
$DATA_IN_REG_MODULE_NAME      = $$arg{name} . "_dr"; # _data_in_reg
```

```
#####
```

```
# Conditional file list.
```

```
#
```

```
# If we're building the whole core (HDL files), then
# we run Vpp on a whole bunch of files. If, on the other hand,
# we're only making the PTF-file, then we run Vpp on many fewer
# files.
```

```
#
```

```
my @vpp_file_list = ();
```

```
if (uc($PTF_ONLY) eq "YES")
```

```
{
```

```
  # Only enough stuff to build the PTF-file:
```

```
  push (@vpp_file_list, (
    "-H", "$VPP_DIR/process_sdf.vpp",
    "-H", "$NIOS_CRYPT_DIR/cpu_interface.vpp",
  ))
};
```

```
} else {
  &Create_Firm_Flip_Flop_Variant ($ADDRESS_OUT_REG_MODULE_NAME,
    $QUARTUS_PROJECT_DIR,
    $part_type);
```

```
  &Create_Firm_Flip_Flop_Variant ($CONTROL_OUT_REG_MODULE_NAME,
    $QUARTUS_PROJECT_DIR,
    $part_type);
```

```
  &Create_Firm_Flip_Flop_Variant ($DATA_IN_REG_MODULE_NAME,
    $QUARTUS_PROJECT_DIR,
    $part_type);
```

```
  push (@vpp_file_list, (
    "-H", "$NIOS_CRYPT_DIR/firm_flip_flop.vpp",
    "-H", "$VPP_DIR/generator_functions.vpp",
    "-H", "$VPP_DIR/process_sdf.vpp",
    "-H", "$NIOS_CRYPT_DIR/processor_generator_functions.vpp",
    "-H", "$NIOS_CRYPT_DIR/cpu_interface.vpp",
    "-H", "$NIOS_CRYPT_DIR/mnemonics.vpp",
    "-H", "$NIOS_CRYPT_DIR/control_bits.vpp",
    "$NIOS_CRYPT_DIR/major_opcode_table.vpp",
    "$NIOS_CRYPT_DIR/subtable_w.vpp",
    "$NIOS_CRYPT_DIR/instruction_decoder.vpp",
    "$NIOS_CRYPT_DIR/cpu_core.vpp",
    "$NIOS_CRYPT_DIR/register_ram.vpp",
  ))
};
```

```
if ($$arg{multiply} =~ /YES/i) {
  push (@vpp_file_list, (
    "$NIOS_CRYPT_DIR/mul_unit.vpp",
  ))
};
```

```
}
}
```

```
# First, run Vpp to create the CPU and all its peripherals:
```

09680106 "061201

```

&Vpp (
    split (/s+/, "-Q -R -X v"),
    "-D", "$QUARTUS_PROJECT_DIR",
    "-P", "$arg{name} . "_",
5    "NIOS_DATA_BITS = $$arg{bits} ",
    "NIOS_SINGLE_CLOCK_SHIFT_DEPTH = $$arg{shift_size}",
    "NIOS_REGISTER_FILE_SIZE = $$arg{num_regs} ",
    "NIOS_WRITEABLE_WVALID_REGISTER = $$arg{wvalid_wr} ",
    "NIOS_MSTEP_SUPPORT = $$arg{mstep} ",
10    "NIOS_MULTIPLY_SUPPORT = $$arg{multiply} ",
    "NIOS_USER_INSTRUCTION_LIST = $$arg{user_instruction_list}",
    "NIOS_VECBASE = $$arg{vecbase} ",
    "NIOS_RESET_ADDRESS = $$arg{reset_a} ",
    "NIOS_SYSTEM_HIGHEST_ADDRESS = $$arg{high_a} ",
15    "NIOS_TURNAROUND_IDLE_CYCLE = $$arg{idle_cycle}",
    "NIOS_MAIN_MEM_BASE_ADDRESS = $$arg{mm_base} ",
    "NIOS_MAIN_MEM_ADDRESS_SPAN = $$arg{mm_span} ",
    "NIOS_PTF_ONLY = $PTF_ONLY ",

20    "MAKE_PTF = TRUE",
    "PTF_FILENAME = $$arg{name}.ptf",
    "PTF_PARAMETER_STRING = $named_arg_string",
    "PTF_PARAMETER_SECTION_NAME = WIZARD_SCRIPT_ARGUMENTS",

25    @vpp_file_list,
);

#####
# If we're in "PTF-only" mode, then we
# also do the user the courtesy of generating a "nios.v" file
# which says "Sorry I didn't -really- build you a Nios core--I was
# just kidding."
#
30    if (uc($PTF_ONLY) eq "YES")
    {
        # Note: "nios.vpp" isn't actually encrypted (it's important
        # that the user sees the comments!).
        #
        &Vpp ("-Q", "-X", "v",
40            "-O", "$QUARTUS_PROJECT_DIR/$$arg{name}.v",
            "NIOS_CPU_NAME = $$arg{name} ",
            "$NIOS_CRYPT_DIR/nios.vpp",
            );

45    warn ("VPP HDL-GENERATION SUCCESSFUL.\n");
    return; # Don't do any more if we're in PTF-ONLY mode.
    }

#
50    #####
    # Everything after here gets executed only if we're really
    # building a CPU and all its HDL-files, etc. (not just a ptf-file).
    #

55    # Get "{quartus,leonardo,modelsim}_define.v, and all that stuff:
    #
    &Copy_Tool_Control_Files ($QUARTUS_PROJECT_DIR);

    # Run Vpp again to create the Leo synthesis script.
    #
60    #NO MORE SYNTHESIS SCRIPTS
    #my $script_name = $$arg{name} . "_synthesis_script.tcl";
    #&Vpp ("-Q",

```

0930106 "061201
T02T00"


```

5  #"-O", "$QUARTUS_PROJECT_DIR/$script_name",
   # "NIOS_CPU_NAME" = $$arg{name} ",
   # "NIOS_SETTINGS_SCRIPT_NAME" = $$arg{leo_settings} ",
   # "$NIOS_SOURCE_DIR/cpu_synthesis_script.tcl.vpp",
   #);

   # Make the simulation files for the embedded decoder ROMs.
   #
   # Now we need to convert the mif-file into a .dat-file for simulation:
10  my $table_name = $$arg{"name"} . "_major_opcode_table";
   &Convert_Mif_To_Dat (" $QUARTUS_PROJECT_DIR/$table_name.mif",
                        "$MODELSIM_DIR/$table_name.dat");

15  $table_name = $$arg{"name"} . "_subtable_w";
   &Convert_Mif_To_Dat (" $QUARTUS_PROJECT_DIR/$table_name.mif",
                        "$MODELSIM_DIR/$table_name.dat");

20  warn ("VPP HDL-GENERATION SUCCESSFUL.\n");
   }

1;  # You need this for Perl to agree that this is a module.

```

09880106 061201

pbm_gen.pm

```
#####
# pbm_and_system_modules.vpp
5 #
# This file contains a bunch of VPP-code which generates
# the systems' PBM module and top-level system-module.
#
# It builds the system as-specified by the PTF-file you
10 # indicate. You do so indicate
#
#   Generate_System_Logic
#
# This one happy Perl function is the top-level call for
15 # creating synthesizable Verilog which implements the
# "system" (e.g. Nios system) described by a single
# PTF-file.
#
# The one-and-only argument is, of course, the name
20 # of the PTF-file itself.
#
# The result is a bunch of Verilog (and other) files
# deposited in the users' Quartus project directory.
#
25 # Naturally, &Generate_System_Logic is a pretty complex
# Perl function which relies upon various dedicated Perl subroutines
# and utilities to do its many jobs. Those sub-functions are
# also defined in this module.
#
30 #####

use ptf_update;
use mk_bsf;

35 #####
# Things to document:
#
# Philosophy, including overview of data-structure
#
40 # difference between "active" and "asserted"
#

#####
# Notes for my friends:
45 #
# Add to system-level WIZARD_SCRIPT_ARGUMENTS section:
#   "Principal_Tri_State_Data_Bus"
#
# Add to module-level SYSTEM_BUILDER_INFO sections:
50 #   "Uses_Registered_Select_Signal"
#   "Uses_Tri_State_Data_Bus"
#
# Master's SBI needs:
#   "Data_Width"
55 #   "Address_Width"
#
# Narrow accesses from non-dynamic tri-state peripherals are filled
# with "undefined." Sorry. That's life.
60

#####
# Funny Things to test:
```

09880106-061201

```

#
#   System with absolutely zero wait-states.
#
#   Systems with -all- tri-state peripherals.
5  #

#####
## ERRORS TO CHECK:
10 #
#   Funny chip-selects (base not a multiple of span, span not power of two).
#
#   Warn about unknown assignments in PTF/SDF file
#
15 #   Check assignments as much as possible when PTF/SDF is parsed,
#   give line numbers.
#
#   Check to be sure all "instances" have unique names.

20 #####
# numerically, a subroutine that enables us to sort by numeric order instead
# of alphabetic order.
#####
sub numerically {$a <=> $b;}

25 sub Find_Max
{
    my $max = "";
    foreach $val (@_)
30     {$max = $val if $val > $max || $max eq "";}
    return $max;
}

#####
35 # &Debug
#
# First arg turns message(s) on/off.
#
#####
40 sub Debug
{
    my ($doit, @messages) = (@_);

    return if !$doit;
    my $space = "";
    foreach $msg (@messages)
45     {
        print STDERR "DEBUG: $space$msg\n";
        $space = "    ";
50     }
}

#####
55 # PBM_Progress
#
# Our own private progress-printing routine.
# Uses the one in "wiz_utils.pm," except that it
# qualifies the output so that it only shows-up if the
# PBM_VERBOSE flag is on, and if this is pass 2.
#
60 #####
sub PBM_Progress
{

```

```

my $old_out = select(STDOUT);
&Progress (@_);
select ($old_out);
}
5
#####
# PTF_Err
#
# In the future, I'd like to put a little more informative
10 # gingerbread around error messages--like the line number and such.
# This'd be the place to do it.
#
#####
sub PTF_Err
15 {
    my $msg;
    ($msg) = (@_);

    die ("Error while reading PTF file.\n
20     $msg");
}

#####
# Get_Port_By_Role
#
# Simple utility-function for acting on %Mod-hashes.
# You say what "avalon role" you want, and this returns
# a %Port-hash (by reference) for the corresponding module port.
#
30 # If there's more than one port with the avalon-role you asked for,
# then you get one of them at-random. This is only reliable if the
# avalon-role uniquely describes one of the module's ports.
#
# If you set the "$strict" option, you get an error if the port
35 # doesn't exist.
#
#####
sub Get_Port_By_Role
40 {
    my ($Mod, $avalon_role, $strict) = (@_);

    # Do step-by-step hash-dereferencing using temporary variables--
    # otherwise the syntax gets impenetrable.
    #
45 my $avalon_port_table = $$Mod{avalon_port_table};
my $Port = $$avalon_port_table{$avalon_role};

    die "Get_Port_By_Role: No port of type $avalon_role on module $$Mod{name}."
    if $strict && !$Port;

50    return $Port;
}

#####
55 # Get_Sys_Signal
#
# This is a utility-function that works on one of our
# defined-by-convention %Mod-hashes (which, you certainly remember,
# is a hash containing various useful information about each module)
#
60 # In particular, each module has a bunch of ports, some of which
# have an "avalon role." Any port with an "avalon role" connects
# to some ritualistically-named signal at the system level.

```

```

#
# Sometimes, given a module and an avalon role, it is useful to
# know what system-level signal "serves" that role.  As an example,
# it is useful to be able to answer the question:
5
#
# -- Which system-level signal drives the "address"-type port
# on the module "Uart_3?"
#
# This function here answers that very sort of question by
10
# digging the appropriate information out of the %Mod-hash you
# pass-in (by reference, of course).
#
# If you set the "$strict"-option argument, then we complain if
# no such port is found on the indicated module.  Otherwise, we
15
# return "" (null) for nonexistent ports.
#
#####
sub Get_Sys_Signal
20
{
    my $Port = &Get_Port_By_Role (@_);

    return "" if !$Port;    # Explicit null-string return if port doesn't exist.

    return $$Port{system_signal};
25
}

#####
# Get_Sys_Module_List
#
30
# This is a utility-function that works on one of our
# defined-by-convention %Sys-hashes (which, you certainly remember,
# is a hash containing various useful information about the system-
# under-construction).
#
35
# This returns a list of %Mod-hash refs.  This list includes
# -all- modules in the system, including the master.
#
# The astute reader will note that a call to this function
# can be replaced by a single expression:
40
#
#     values %{$$Sys{module_table}}
#
# But that's one ugly expression.  This function adds a bit
# of self-documentation, and a much-needed dose of object-oriented,
45
# implementation-hiding, pseudo-access-method methodology.
#
#####
sub Get_Sys_Module_List
50
{
    my ($Sys) = (@_);

    my $module_table = $$Sys{module_table};    # Just for nice syntax.
    return values (%$module_table);
55
}

#####
# Get_Sys_Slave_List
#
# Just like "Get_Sys_Module_List," above, except that the list
60
# you get doesn't include the master.
#
# Many times, this is what you really wanted.  And this function
# saves you the trouble of having to put a master-exclusion test

```

```

# in your otherwise-tidy loop.
#
#####
5  sub Get_Sys_Slave_List
    {
        my ($Sys) = (@_);

        my @result = ();

10     foreach $Mod (&Get_Sys_Module_List($Sys))
        {
            push (@result, $Mod) unless $$Mod{Is_Bus_Master};
        }

15     return @result;
    }

#####
# Get_Module_Port_List
20 #
# Just like "Get_Sys_Module_List," above-- and intended to
# serve the same dubious code-beautification purpose.
#
# The difference is: this function gets all the %Port-hashes
25 # out of a %Mod-hash, instead of all the %Mod-hashes out of a
# %Sys-hash.
#
#####
30 sub Get_Module_Port_List
    {
        my ($Mod) = (@_);

        my $port_table = $$Mod{port_table}; # Just for nice syntax.
        return values (%$port_table);

35     }

#####
40 # Emit_Comment
#
# Emit the user-supplied string as a verilog comment directly
# into the output file. As a courtesy, we put //-characters
# at the beginning of every line, sparing the user the trouble.
45 #
# We call "&Vprint," so the user must previously have "selected"
# their destination verilog-file as STDOUT.
#
#####
50 sub Emit_Comment
    {
        my ($comment, $language) = (@_);
        $language = "verilog" if !$language;

55     my $cstart = "-- ";
        $cstart = "// " if $language =~ /^verilog/i;

        $comment = $cstart.$comment;
        $comment =~ s|\n|\n$cstart|mg; # Put '// ' in front of every line.

60     &Vprint ("$comment\n");
    }

```

09:30:10 "061201

```
#####
# Emit_Top_Comment
#
# Writes module-name, date-stamp, and legal notice into the
5 # currently-selected output file, trilingually.
#
#####
sub Emit_Top_Comment
10 {
    my ($module_name, $lang) = (@_);

    $lang = "verilog" if !$lang;

    my $date = scalar (localtime());
15    my $magic_altera_string = '%Altera Excalibur Nios(tm)%';
    my $stop_comment=<<EOM;
    // megafunction wizard: $magic_altera_string
    //// GENERATION: STANDARD
    //// VERSION: WM1.0
20    // Module: $module_name
    //
    // Automatically-generated file: **** DO NOT EDIT ****
    // Generated by Excalibur SOPC-Builder    [$date]
    //
25 $GLOBAL_COPYRIGHT_NOTICE
    //
    EOM

    $stop_comment =~ s|//|--|mg if $lang =~ /^(vhdl|ahdl)/i;
30    &Vprint ($stop_comment);
}

#####
# Emit_Module_Header
35 #
# You give the name of the module ("Foo"), and this function
# emits the ritualistic top-of-module stuff, which we once would have
# done this way:
#
40 #     module Foo ( /*{&Declare_Ports_For("Foo")}*/ )
#                   /*{ &Define_Ports_For("Foo")}*/
#
# Emits the module- and port-delcarations for the named module into
# the currently-selected output file.
45 #
# Naturally, you have to have already done a &List_Ports_For the named
# module.
#
#####
50 sub Emit_Module_Header
{
    my ($module_name, $lang, $do_bb_declaration) = (@_);

    $lang = "verilog" if !$lang;

55
    if (($lang =~ /^(vhdl/i)) {
        &Vprint ("LIBRARY ieee;\n");
        &Vprint ("use ieee.std_logic_1164.all;\n");
60        &Vprint ("ENTITY $module_name IS\n");
        &Define_Ports_For ($module_name, 0, $lang);
        &Vprint ("END $module_name;\n");
        &Vprint ("ARCHITECTURE behavior OF $module_name IS\n");
```

```

} elsif ($lang =~ /^ahdl/i) {
    &Vprint ("SUBDESIGN $module_name\n");
    &Vprint ("(\n");
5    &Define_Ports_For ($module_name, 0, $lang);
    &Vprint (")\n");

} elsif ($lang =~ /^verilog/i) {
    # Verilog:
10    my $synplify_bb_string = '/* synthesis syn_black_box */'
        if $do_bb_declaration;

    &Vprint ("module $module_name (\n");
    &Declare_Ports_For ($module_name);
15    &Vprint (") $synplify_bb_string ;\n");
    &Define_Ports_For ($module_name, 0, $lang);
    &Vprint ("\n // synopsys translate_off \n")
        if $do_bb_declaration;
    &Vprint ("\n");
20    } else {
        die "Emit_Module_Header: Foul language ($lang)";
    }
}

25 #####
# Emit_VHDL_Component
#
# You must declare all your VHDL black-boxes ("COMPONENT"s) in
30 # the "ARCHITECTURE" section of your module, before the first
# "BEGIN." OK, so be it.
#
# You must have previously done a &List_Ports_For -call for this
# module.
35 #
#####
sub Emit_VHDL_Component
{
    my ($comp_name) = (@_);
40    &Vprint ("COMPONENT $comp_name IS\n");
    &Define_Ports_For ($comp_name, 0, "vhdl");
    &Vprint ("END COMPONENT;\n");
}

45 #####
# PBM_Assign
#
# Emit a simple assignment into the PBM-file (which we presume
# to be the currently-selected output file).
50 #
# We take special measures to avoid redundant assignments (we
# keep our own private hash of past assignments). We check to make
# sure the same signal is never assigned to two different things.
#
55 # The arguments are the target-signal and the value we want
# assigned to it.
#
# Also, note that we -explicitly- do nothing if the assignment-target
# is NULL (""). This deals gracefully with, for example, broadcast-
60 # assignments to modules that don't have a recipient port. For example,
# you can go ahead and &PBM_Assign the write-data bus to a module that
# doesn't actually have a "writedata"-type port, and it still works
# out OK (nothing happens).

```



```

#
#####
%__Private_PBM_Assign_Hash__
sub PBM_Assign
5  (
    my ($target_signal, $assignment_value) = (@_);

    return if $target_signal eq "";    # Deal with null-assignment, per comment.

10  my $previous_assignment = $__Private_PBM_Assign_Hash__{$target_signal};

    # Ignore truly-redundant assignments:
    return if $previous_assignment eq $assignment_value;

15  $previous_assignment eq "" or die "
    Inconsistent assignments to signal $target_signal:
        ($previous_assignment) and ($assignment_value)";

    &Vprint ("assign $target_signal = $assignment_value;\n");

20  $__Private_PBM_Assign_Hash__{$target_signal} = $assignment_value;
    )

#####
25  # PBM_Wire
    #
    # Emit a Verilog "wire" declaration into the currently-selected
    # output file.  You give the name and width of the wire you
    # want to declare.  Deals gracefully with the null wirename ("") and
30  # with zero-width wires:  No declaration is emitted.
    #
    # Also allows you to pass-in an optional "$assignment" Verilog-expression,
    # so you can declare a wire and assign a value to it in one stroke.
    #
35  #####
    sub PBM_Wire
    (
        my ($wirename, $width, $assignment) = (@_);

40        $width = 1 if $width eq "";

        return if $width == 0;
        return if !$wirename;

45        my $range = &W($width);

        &Vprint ("wire $range $wirename;\n") if $wirename && $width;

        &PBM_Assign ($wirename, $assignment) if $assignment;

50    )

#####
    # Validate_And_Reserve_Address_Range
    #
55  # Given a reference to a %Mod-hash, we compute the
    # address-range allocated to that module and make sure that
    # somebody else doesn't already live there.  If so, we
    # print an error.
    #
60  # Also, while we're thinking about the address, we do some sanity-checks.
    # That's why the %Sys-hash (ref) is also passed-in, so we can
    # check to be sure the module is in-bounds.
    #

```

09880106 "061201

09880106 061201
T02150 9018850

```
# We also declare the silly "&within"-function for code-beauty.
#
#####
sub within { my ($test, $lo, $hi); return ($test >= $lo) && ($test <= $hi);}
5 %__Private_Address_Range_Hash__

sub Validate_And_Reserve_Address_Range
{
10 my ($Mod, $Sys) = (@_);

    $$Mod{Base_Address} ne "" or die
        "Error: bad Base_Address setting for module $$Mod{name}";

    $$Mod{address_span} = 2*($$Mod{highest_address_bit_used} + 1);
15 $$Mod{end_address} = $$Mod{Base_Address} + $$Mod{address_span} - 1;

    $$Mod{end_address} <= ($$Sys{address_span} - 1) or die "
        End-address of module $$Mod{name} is greater than maximum system
        address ($$Sys{address_span} - 1)";

20 foreach $inst (keys(%address_range_list))
    {
        my ($min,$max) = split (/\\/, %__Private_Address_Range_Hash__{$inst});

25 die "Address-range conflict between $inst and $$Mod{name}.
            $inst occupies: [$min ...$max]
            $$Mod{name} occupies: [$$Mod{Base_Address} .. $Mod{end_address}]"
        if &within ($$Mod{Base_Address}, $min, $max) ||
            &within ($$Mod{end_address}, $min, $max) ;

30 }

    %__Private_Address_Range_Hash__{$$Mod{name}} =
        "$$Mod{Base_Address},$$Mod{end_address}";

35 }

#####
# Resolve_Address_Alignments
#
# In the PTF-file, the user can specify "dynamic" and "native"
# address-alignments. In these cases, the system-generator (that'd be
40 # me) needs to "do something smart" to reconcile peripherals with
# nonnative data-widths.
#
# We can only do that after all modules have been read-in (including,
45 # significantly, the master), and some system-level info has been
# set-up.
#
# Consequently, this function is called at the end of
# &Get_System_Data_From_PTF, after all the modules have been read.
50 #
#####
sub Resolve_Address_Alignments
{
55 my ($Sys) = (@_);

    foreach $Mod (&Get_Sys_Slave_List($Sys))
    {
        # Figure out what kind of address will ultimately be presented to this
        # module. This is slightly tricky because of the special "dynamic" case,
60 # in which case we have to look at the data width.
        #
        if ($$Mod{Address_Alignment} eq "dynamic")
        {
```

```

# Dynamic alignment: address-type depends on the data size:
$$Mod{address_type_used} =
    $$Mod{Data_Width} > 16 ? "word" :
    $$Mod{Data_Width} > 8 ? "halfword" :
    "byte" ;

} elsif ($$Mod{Address_Alignment} eq "native") {
    $$Mod{address_type_used} =
        $$Sys{master_data_width} == 16 ? "halfword" :
        "word" ;
} else {
    # "Normal" old-style alignment:
    warn ("
        Old-style Address_Alignment ($$Mod{Address_Alignment})
        found for module $$Mod{name}.\n") if $$Sys{verbose};
    $$Mod{address_type_used} = $$Mod{Address_Alignment};
}

# When is a "dynamic" module -not- "dynamic"? When it happens
# to be the same width as the master. Check width here, and set
# a per-module flag which tells us whether to really, truly use
# dynamic bus-sizing for this module.
#
if (($$Mod{Address_Alignment} eq "dynamic" ) &&
    ($$Mod{Data_Width} < $$Sys{master_data_width}))
{
    $$Mod{is_dynamically_sized} = 1;
} else {
    $$Mod{is_dynamically_sized} = 0;
}

# It's nice to know if the system has any dynamic bus-sizing:
$$Sys{has_dynamic_bus_sizing} = 1 if $$Mod{is_dynamically_sized};

# This is a handy thing to know about a module:
$$Mod{highest_address_bit_used} =
    $$Mod{address_type_used} eq "byte" ? ($$Mod{Address_Width} - 1) :
    $$Mod{address_type_used} eq "halfword" ? ($$Mod{Address_Width} ) :
    ($$Mod{Address_Width} + 1) ;
}
}

#####
# Get_System_Data_From_PTF
#
# Given a reference to a mostly-empty %Sys-hash, we read the
# PTF file, build a %Mod-hash for every module and a %Port-hash
# for every port, and stuff the results back into the %Sys
# data structure.
#
# We do as much "peephole" error-checking as we can while
# reading-in ports -- confirming allowed values for PTF fields,
# screening out "impossible" port types, etc.
#
# But there is a certain amount of checking that we -can't- do
# until we've read-in the entire system--checking to be sure no
# module's data bus is wider than the master, for example, has to
# wait until all modules are read-in. Those kinds of checks
# -do not- get executed here.
#

```

09880106.061201
102190" 90T08865

```

# Plus, we don't actually "do anything" with the data.  We do
# only pre-processing on the %Sys-hash, so that it will be
# easier, later on, to do what we need.
#

```

```

5 #####

```

```

my %PBM_HDL_EXTENSION;
$PBM_HDL_EXTENSION {verilog} = "v";
$PBM_HDL_EXTENSION {vhdl}    = "vhd";
10 $PBM_HDL_EXTENSION {ahdl}   = "tdf";

```

```

sub Get_System_Data_From_PTF
{

```

```

15   my ($Sys, $db_Sys) = (@_);

   # A hash of listrefs.  The hash-keys are shared-port names:
   my %shared_port_table;

```

```

20   #keys: shared port-names.  Values: bus-group names.
   my %bus_membership_table;
   my @tri_state_bus_list = ();

```

```

25   # For Perl syntax-niceness, build these hashes up in the
   # module loop, then add them to the %Sys datastructure when
   # we're all done:
   #
   my %module_table; undef %module_table;

```

```

30   #####
   # Module loop
   #
   # Accumulate a hash of direct and derived information about each
   # module.
   # When we're all done, we'll add a reference to this hash to
35   # %Sys.
   #
   my $num_children = &get_child_count ($db_Sys);
   for ($child_index = 0; $child_index < $num_children; $child_index++)
   {
40     my $db_Module = &get_child ($db_Sys, $child_index);
     next if &get_name ($db_Module) ne "MODULE";    # ignore non-modules.

```

```

45     #####
     # Read SYSTEM_BUILDER_INFO section
     #
     # This will form the basis for our own private cache of
     # useful info about this module.  Also, now would be a good
     # time to pre-digest and validate all the settings we read
     # from the PTF file.  By this I mean: Converting "TRUE/FALSE"
50     # strings into testable bits, evaluating numerical expressions,
     # and checking for illegal values.
     #
     my $db_SBI = &PTF_Get_Required_Child_By_Path ($db_Module,
                                                    "SYSTEM_BUILDER_INFO");
55     my $sbi = &PTF_Build_Hash_From_Section ($db_SBI);
     my %Mod = %$sbi;
     $Mod{name} = &get_data ($db_Module);
     $Mod{class} = &PTF_Get_Required_Data_By_Path ($db_Module, "class");
60     &PBM_Progress ("Processing module $Mod{name}.") if $$Sys{verbose};

     &PTF_Check_Bool (\%Mod, "Is_Enabled",
                     next unless $Mod{Is_Enabled};    # Quit early for disabled modules.

```

09330106-061201

```

5  &PTF_Check_Bool (\%Mod, "Instantiate_In_System_Module", 1);
   &PTF_Check_Bool (\%Mod, "Uses_Registered_Select_Signal", 0);
   &PTF_Check_Bool (\%Mod, "Uses_Tri_State_Data_Bus", 0);
10 &PTF_Check_Bool (\%Mod, "Has_IRQ", 0);
   &PTF_Check_Bool (\%Mod, "Is_Bus_Master", 0);
   &PTF_Eval (\%Mod, "Base_Address", "N/A");
   &PTF_Eval (\%Mod, "IRQ_Number", );
   &PTF_Eval (\%Mod, "Address_Width", );
15 &PTF_Eval (\%Mod, "Data_Width", );
   &PTF_Eval (\%Mod, "Read_Wait_States", "peripheral_controlled");
   &PTF_Eval (\%Mod, "Write_Wait_States", "peripheral_controlled");
   &PTF_Eval (\%Mod, "Setup_Time", );
   &PTF_Eval (\%Mod, "Hold_Time", "half_clock");
15 &PTF-Allow (\%Mod, "Address_Alignment", "dynamic", "native",
   "byte", "word", "halfword");

# Name module, if so far unnamed:
$Mod{Instance_Name} = "the_$Mod{name}"
20 if (($Mod{Instance_Name} eq "" ) ||
   ($Mod{Instance_Name} eq "--unknown--" ) );

$module_table{$Mod{name}} = \%Mod; # Put reference into system hash.

25 #####
# If this is the master-module,
# set a system-level variable, and check to see
# that this is the only one.
30 if ($Mod{Is_Bus_Master})
{
   $$Sys{master_name} eq "" or die "
   $$Sys{name} has multiple masters: $Mod{name} and $$Sys{master_name}";

   $$Sys{master_name} = $Mod{name};
35   $$Sys{master} = \%Mod;
}

$$Sys {has_registered_select_signals} = 1
   if $Mod{Uses_Registered_Select_Signal};

40 $$Sys {has_tri_state_data_busses} = 1
   if $Mod{Uses_Tri_State_Data_Bus};

45 #####
# Port Loop
#
# Look at each port on this module. Build-up a hash
# of useful information. For most "normal" ports, we
50 # can add a corresponding port on the PBM and/or the system.
# Shared ports get recorded in a hash, so we can
# add them to the system/PBM later, after all the port data
# has been gathered for all modules.

55 if (&get_child_by_path($db_Module, "PORT_WIRING/TYPE")) {
   warn ("pbm_gen: old-style PORT_WIRING section for $Mod{name}.")
   if $$Sys{verbose};
   &PTF_Update_Port_Wiring_Section ($db_Module);
60 }

my $db_Port_Wiring = &get_child_by_path($db_Module, "PORT_WIRING");

```

```

# Hashes which are included, by reference, in the %Module
# data structure. We build them up as temporary variables
# and stick them into the %Mod-hash at the end--otherwise,
# the Perl dereferencing syntax becomes impenetrable:
5  #
my %avalon_port_table = ();
my %port_table        = ();

10 my $num_ports = &get_child_count ($db_Port_Wiring);
for ($port_index = 0; $port_index < $num_ports; $port_index++)
{
    my $db_Port = &get_child ($db_Port_Wiring, $port_index);
    next unless &get_name($db_Port) eq "PORT";

15     # Whatever the PTF says about this port, we want to know.
    my $Port      = &PTF_Build_Hash_From_Section ($db_Port);
    $$Port{name}  = &get_data ($db_Port);
    my $who_died  = "$$Port{name} on module $Mod{name}"; # for errors.

20     # Test some basic stuff:
    $$Port{direction} =~ /^(input|output|inout)$/ or
        die "$who_died: Bad direction '$$Port{direction}'";

    # Port-record has pointer to module parent, and module
    # record has table of all Port-records:
25     $$Port{parent}      = \%Mod;
    $port_table{$$Port{name}} = $Port;

    &PBM_Progress (" Processing port $$Port{name}.") if $$Sys{verbose};

30     # Ports -used to- have scopes. Now we can infer their scope
    # from the module's (and port's) other attributes.
    if ($$Port{scope}) {
        warn ("obsolete port '$$Port{name}' (has 'scope').\n")
            if $$Sys{verbose};

        $$Port{scope} =~ /^(internal|external|master)$/ or
            die "$who_died: Bad scope '$$Port{scope}'";

40         $$Port{is_external} = $$Port{scope} eq "external"; # Testable bool.
    }

    # Decide whether this port is esxternal or internal. We used
    # to require that the user tell us, but now we can figure it
    # out for ourselves:
45     #
    if (($Mod{Instantiate_In_System_Module})) {
        # For modules -inside- the system-module, all their
        # avalon-ports are internal. All their non-avalon ports
        # are external
50         $$Port{is_external} = 1 if !$$Port{avalon_role};
    } else {
        # For modules -outside- the system-module, all their
        # avalon-ports are external. All their non-avalon ports
        # are none of our business
55         $$Port{is_external} = 1 if $$Port{avalon_role};
    }

60     # Some consistency-checking:
    # * Only external signals can be "shared":
    # * All internal (or master) signals must have an avalon-role.
    die "$who_died: only external ports may be shared."
}

```

00000106.061201
T02T90" 90T08860

```

        if ( $$Port{is_shared}    && !$$Port{is_external} );

#####
# What does this port connect to?
5  #
# For a "typical" system-internal module with nothing "shared,"
# each port gets connected to a unique, dedicated wire in the
# system module.  That wire will go to one of two places:
10 # a like-named port on the PBM (for avalon signals) or be
# promoted to a system-level port. Easy enough.
#
# There are two wrinkles to consider: modules which are -not-
# instantiated in the system, and modules which have shared
# ports.
15 #
# *** External Modules
#
# If the module is not instantiated inside the system, then
# we do twp special things:
20 #
#     1) Ignore any signals which don't have an avalon_role,
#         because they're not our responsibility, anyhow.
#
#     2) Promote its PBM-ports to system-level I/Os with the
#         same direction and name.
25 #
# **** Shared Ports
#
# Ports are only "shared" if they're part of a tri-state
# bus structure.  All tri-state busses are named.  Here
30 # are some examples of system-level signals which are part
# of a shared tri-state bus:
#
#         memory_bus_address
#         memory_bus_byteenablen
35 #
#         ide_data
#         ide_writen
#
# In this case, the signal names are a concatenation of the
# tri-state bus group name and the avalon role.  We don't
# add these ports to the PBM/system -yet-, because we don't
# really know their widths until all the modules have been
# processed.  Instead, we just record our %Port as a "client"
40 # of this shared port in a hash.  Later, we'll work out
# exactly how the shared ports show up on the system module.
#
if ($$Port{is_shared})
{
50     die "
        Shared port $$Port{name} on module $$Mod{name} has no
        'Avalon role'
        if (!$$Port{avalon_role});

55     die "
        Shared port $$Port{name} found on module $Mod{name}, but
        module does not use tri-state data bus"
        if ((!$Mod{Uses_Tri_State_Data_Bus} ) ||
            ( $Mod{Tri_State_Data_Bus} eq "" ) );

60     my $shared_port = "$Mod{Tri_State_Data_Bus}_$$Port{avalon_role}";
    $$Port{system_signal} = $shared_port;

```

```

# Record the fact that this %Port is a client of
# this shared port. Push a reference to this %Port-hash
# onto this shared-port client list:
#
5   push (@{$shared_port_table{$shared_port}}, $Port);

# also record which tri-state bus group this shared port
# belongs to. (true, you could figure it out by looking
# at it's name, but that just sounds risky to me:
10  #
    $bus_membership_table{$shared_port} = $Mod{Tri_State_Data_Bus};
} else {

    # This is -not- a shared port, so we make one of those
15  # much-beloved machine-generated port names
    # (e.g. bidir_port_to_and_from_the_lcd_pio).
    #
    my $transfer_str = ($$Port{direction} eq "input") ? "to" :
                       ($$Port{direction} eq "output") ? "from" :
20                      "to_and_from";

    $$Port{system_signal} =
        "$$Port{name}_$transfer_str\_$_Mod{Instance_Name}";
}

25  # Construct an inverse hash so we can look-up ports
    # -by avalon role- for this module:
    #
    $avalon_port_table {$$Port{avalon_role}} = $Port
30     if $$Port{avalon_role};

#####
# Record some useful system-level and
# module-level information as the ports
# go by:
35  #
    $$Sys{master_address_width} = $$Port{width}
        if ($Mod{Is_Bus_Master} ) &&
        ($$Port{avalon_role} eq "address" ) ;

40  $$Sys{master_data_width} = $$Port{width}
        if ($Mod{Is_Bus_Master} ) &&
        ($$Port{avalon_role} eq "writedata" ) ;

45  } #end: %Port-loop

# Attach the hashes we built-up in the %Port-loop into the
# %Mod data structure:
#
50  $Mod{port_table} = \%port_table;
    $Mod{avalon_port_table} = \%avalon_port_table;

#####
# Derived Module Info
55  #
    # Pre-digest some useful facts about this module:
    #
    # It's nice to have a list of all tri-state busses in the system:
    push (@tri_state_bus_list, $Mod{Tri_State_Data_Bus})
60     if $Mod{Uses_Tri_State_Data_Bus};

# Predigest hold-time values: handle "half-clock" case.
$Mod{hold_time_full_clocks} = $Mod{Hold_Time};

```

0900106 "061201


```

    "clk" | input | input | \ $W == 1
\ $W = 1,
    resetn | input | input | \ $W == 1
5 \ $W = 1,
    always0 | input | input | \ "Any width OK\"
\ $W = "\"",
    always1 | input | input | \ "Any width OK\"
\ $W = "\"",
10 \ $W = 1,
    writen | input | output | \ $W == 1
    readn | input | output | \ $W == 1
\ $W = 1,
    byteenablen | input | output | \ $W <= 4
15 \ $W = 4,
    waitrequest | output | input | \ $W == 1
\ $W = 1,
    irq | output | input | \ $W == 1
\ $W = 1,
    irqnumber | N/A | input | \ $W <= 6
20 \ $W = 6,
    chipselect | input | N/A | \ $W == 1
\ $W = 1,
    registeredselectn | input | N/A | \ $W == 1
\ $W = 1,
25 ifetch | N/A | output | \ $W == 1
\ $W = 1,
    memis32bits | N/A | input | \ $W == 1
\ $W = 1,
    data | inout | N/A | \ $W <= \ $\$Sys{master_data_width}
30 \ $W = \ $\$Sys{master_data_width},
    readdata | output | input | \ $W == \ $\$Mod{Data_Width} &&
    \ $W <= \ $\$Sys{master_data_width}
\ $W = \ $\$Mod{Data_Width},
35 writedata | input | output | \ $W == \ $\$Mod{Data_Width} &&
    \ $W <= \ $\$Sys{master_data_width}
\ $W = \ $\$Mod{Data_Width},
40 address | input | output | \ $W > 0 &&
    \ $W <= \ $\$Sys{master_address_width}
\ $W = \ $\$Mod{Address_Width},
";
# Turn our pretty text-table into a bunch of code-useful hashes:
45 #
#now some hashes;
my %required_slave_dir;
my %required_master_dir;
my %width_requirement;
50 my %width_default;

$requirement_table =~ s/\s+//sg;
foreach $req (split (/\\s*\\,\\s*/s, $requirement_table))
{
55 my ($role, $slave_dir, $master_dir, $width_condition,
    $width_default_string) =
        split (/\\s*\\|\\s*/s, $req);

    $required_slave_dir {$role} = $slave_dir unless $slave_dir eq "N/A";
60 $required_master_dir {$role} = $master_dir unless $master_dir eq "N/A";
    $width_requirement {$role} = $width_condition;
    $width_default {$role} = $width_default_string;
}

```

```

my %return_hash;
$return_hash{required_slave_dir} = \%required_slave_dir;
$return_hash{required_master_dir} = \%required_master_dir;
5  $return_hash{width_requirement} = \%width_requirement;
   $return_hash{width_default} = \%width_default;

   return (\%return_hash);
}

10 #####
   # Check_Avalon_Rules
   #
   # Given a (the?) system-hash, this function loops over the
15 # data structure module-by-module and port-by-port and checks
   # that no system-level Avalon rules have been violated.
   #
   # These are rules like: All "chipselct"-type ports on a module must
   # be 1 bit wide, all "address"-ports must be inputs, and all
20 # data ports must be narrower than the masters'.
   #
   # This function replaces the section of inline-code formerly known
   # as "The Parade of the Port Types."
   #
25 #####
   sub Check_Avalon_Rules
   {
       my ($Sys) = (@_);

30       my $requirement_hash_table = &Get_Avalon_Requirement_Table;

       my %required_slave_dir = %{$requirement_hash_table->{required_slave_dir}};
       my %required_master_dir = %{$requirement_hash_table->{required_master_dir}};
       my %width_requirement   = %{$requirement_hash_table->{width_requirement}};
35       my %width_default      = %{$requirement_hash_table->{width_default}};

       # Now consider each avalon-port on each module
       # and see if it lives up to our expectations.
       # If port is not defined, set it to default value.

40       foreach $Mod (&Get_Sys_Module_List($Sys))
       {
           my $avalon_port_table = $$Mod{avalon_port_table};
           foreach $role (keys (%$avalon_port_table))
           {
45               {
                   my $Port = $$avalon_port_table{$role};
                   my $W     = $$Port{width};

                   my $required_dir = $$Mod{Is_Bus_Master} ? \%required_master_dir :
50                                     \%required_slave_dir ;

                   #set default width
                   eval ($width_default{$role})
                       if ($W eq "");

55                   #set default direction
                   $$Port{direction} = $required_dir{$role}
                   if ($$Port{direction} eq "");

60                   die "
                       Illegal avalon-role : $role
                       for port $$Port{name} on module $$Mod{name}."
                       if (!$required_dir{$role});

```

```

die "
Illegal direction: $$Port{direction}.
Expected: $$required_dir{$role}
for port $$Port{name} on module $$Mod{name}."
if ($$Port{direction} ne $$required_dir{$role});

my $width_ok = eval ($width_requirement{$role});
die "Bad check-expression: $width_requirement{$role} ($@)" if $@;

die "
Illegal width: $W
for port $$Port{name} on module $$Mod{name}."
if (!$width_ok);

#####
# Per-port Consistency-checks
#
# Does this port agree with things that were said in the
# module's SBI section?
#
if ($role =~ /data$/) {
    $$Mod{Data_Width} == $$Port {width} or die "
Width of data port $$Port{name} ($$Port{width}) is inconsistent
with 'Data_Width' setting for module $$Mod{name}.";
}

if ($role eq "address") {
    $$Mod{Address_Width} == $$Port {width} or die "
Width of address port $$Port{name} ($$Port{width}) is
inconsistent with 'Address_Width' setting for
module $$Mod{name}."
}
}

# Half-clock hold-time only allowed with zero setup-time:
if (($$Mod{Hold_Time} =~ /half/)) {
    $$Mod{Setup_Time} == 0 or die "
Error in module $$Mod{name}: half-clock 'Hold_Time' setting
allowed only if 'Setup_Time' is 0 (zero).";
}

next if $$Mod{Is_Bus_Master}; # The following checks are for mere slaves.

#####
# Funny Setup/Hold Rule
#
# Here's the rule:
#
# -- If you have a nonzero hold-time, then you get a nonzero
# setup-time, whether you asked for one or not.
#
# Here's the explanation:
#
# You may recall that the masters' readn- and writen- signals come
# directly from the Q-output of registers. Timing-wise, this is
# a good and happy thing. Peripherals which don't explicitly request
# setup/hold times get the masters' registered strobe signals, and
# life is good.
#
# But now consider the plight of the poor module who requests
# setup/hold time. Necessarily, he now gets his own customized

```

```

# (narrower) version of the read- and write-strobes. It would be
# oh-so-nice if his custom strobes also came directly from Q-outputs
# of registers. (And, I note, it would also make the
# strobe-customization logic easier for your humble implementor).
5 # So now we have the custom readn- and writen-strobes coming from
# register outputs. But, obviously, this happy register introduces
# a one-clock delay between the time we decide to assert the custom
# strobe and the time it gets asserted. That's fine--we have
# (at least) one clock-cycle of slack before we need to assert the
10 # strobe--as long as there's a (nonzero) setup-time. That gives
# us the clock we need to "customize" and still have a register.
# So. Phrased concisely:
#
# Custom strobes want registers
# registers imply delay
# nonzero setup-time accomodates delay
#
# --> custom strobes need a nonzero setup-time.
#
20 # We print a warning when we encounter such a case.
#
if (($Mod{Setup_Time} == 0) &&
    ($$Mod{hold_time_full_clocks} > 0)
    )
25 {
    $$Mod{Setup_Time} = 1;
    warn ("
        Setup-time (1 clock) automatically added for module $$Mod{name}.
        Modules with nonzero hold-time automatically get at least
30         one clock of setup-time.\n
    ");
}

# Some extra module-level consistency-checking between
# "System Builder Info" and port-types.
#
&Validate_And_Reserve_Address_Range ($Mod, $Sys);

40 die "Module $$Mod{name} 'Has_IRQ', but no pin is of type 'irq'."
    if $$Mod{Has_IRQ} && !$Savalon_port_table{irq};

die "Module $$Mod{name} has an irq-pin, but 'Has_IRQ' is FALSE."
45 if !$Mod{Has_IRQ} && $$Savalon_port_table{irq};

die "Module $$Mod{name} has word-alignment, but master is not 32 bits."
50 if ($$Mod{address_type_used} eq "word" &&
    $$Sys{master_data_width} < 32
    );

die "Module $$Mod{name}: Data is too wide for dynamic alignment."
55 if ($$Mod{is_dynamically_sized} &&
    $$Mod{Data_Width} > ($$Sys{master_data_width} / 2) );

die "Module $$Mod{name}: must set SBI/Uses_Registered_Select_Signal."
if (!$$Mod{Uses_Registered_Select_Signal} &&
    $$Savalon_port_table{registeredselectn});

60 die "Module $$Mod{name}: peripheral controlled wait \n".
    "not supported for registered chip selects\n"
    if ($$Mod{Uses_Registered_Select_Signal} &&
        (($$Mod{Read_Wait_States} =~ /peripheral_controlled/i) ||
        ($$Mod{Write_Wait_States} =~ /peripheral_controlled/i))
    );

```

09880106 "061201

```

#hold time means theres a setup time,
die "Module $$Mod{name}: peripheral controlled wait not\n".
    "supported for peripherals with non-zero setup and/or hold times\n"
5     if (($Mod{Setup_Time}) &&
        (($Mod{Read_Wait_States} == /peripheral_controlled/i) ||
        ($Mod{Write_Wait_States} == /peripheral_controlled/i))
        );

10    die "Module $$Mod{name}: can't find 'registeredselectn'-type port."
        if ($$Mod{Uses_Registered_Select_Signal} &&
            !$savalon_port_table{registeredselectn});

    my $Port_writen = $$savalon_port_table{writen};
15    die "Module $$Mod{name}: shared writen port illegal if setup/hold > 0."
        if ($$Port_writen{is_shared} &&
            ($$Mod{Setup_Time} + $$Mod{hold_time_full_clocks} > 0) );

    my $Port_readn = $$savalon_port_table{readn};
20    die "Module $$Mod{name}: shared readn port illegal if setup/hold > 0."
        if ($$Port_readn{is_shared} &&
            ($$Mod{Setup_Time} + $$Mod{hold_time_full_clocks} > 0) );

    # ... Add more, please...
25    # Check:
    # Uses_Registered_Select_Signal vs. actual "registeredselectn" ports.
    # Uses_Tri_State_Data_Bus vs. actual shared/data ports.
    # if principal bus declared, system -should- have tri-state busses.
    # principal tri-state bus must be as wide as CPU.
30    # if we have principal bus, at least one module must be on it.
    # You may -not- have shared readn/writen signals with nonzero
    # setup/hold times.
    # system data width should be only -exactly- 16 or 32.

35    } # End: $Mod-loop
}

#####
# Create_System_Port_Lists
40 #
# Given a reference to a (the?) system-hash, this function
# builds a "List_Ports_For-" definition of the system-module
# and the PBM.
#
45 # For the most part, this is easy: We look through the list of all ports
# on all modules. If any have an avalon-role, then their complement
# shows up on the PBM. If any are "external", then they show up on
# the system module.
#
50 # The one nasty little wrinkle is shared-ports. We have kept a separate
# record of all shared ports, and now we use it to compute the appropriate
# width--then we can add the shared ports to both the PBM and system modules.
#
# And then shared ports have one more trick: The width of shared address
55 # ports. For most shared ports, the width is just the width of the largest
# client. For address ports, we need to take -alignment- into account.
#
#####
sub Create_System_Port_Lists
60 {
    my ($Sys) = (@_);

    # The clk and reset_n ports are magic. They -always- appear

```

on both the system and the PBM.

```
#
$$Sys(pbm_list_ports_for_string) = "clk      | 1 | input,
reset_n | 1 | input,";
5  $$Sys(system_list_ports_for_string) = $$Sys(pbm_list_ports_for_string);
```

```
foreach $Mod (&Get_Sys_Module_List($Sys)) {
  foreach $Port (&Get_Module_Port_List ($Mod))
```

```
{
  next if $$Port{is_shared};    # Do shared ports later.
```

```
  if ($$Port{avalon_role})
  {
    my $pbm_port_dir = $Complementary_Direction {$$Port{direction}};
    my $pbm_port_description =
      "$$Port{system_signal} | $$Port{width} | $pbm_port_dir,";
```

```
    $$Sys(pbm_list_ports_for_string) .= $pbm_port_description;
    $$Sys(system_list_ports_for_string) .= $pbm_port_description
    if (!$$Mod{Instantiate_In_System_Module});
  } else {
```

```
    # OK, this port doesn't have an avalon role. For
    # externally-instantiated modules, we just ignore it.
    # for internally-instantiated modules, we promote it
    # to a top-level system port with the -same- direction
    # as the port on the module.
```

```
    #
    # All non-avalon-type ports must be external
    $$Sys(system_list_ports_for_string) .=
      "$$Port{system_signal} | $$Port{width} | $$Port{direction},"
      if ($$Mod{Instantiate_In_System_Module});
```

```
  }
} # End: $Port-loop
```

```
} # End: $Mod-loop
```

```
#####
```

```
# Now do the shared ports.
```

```
#
40 my $shared_port_table = $$Sys(shared_port_table);
```

```
foreach $shared_port_name (keys(%$shared_port_table))
{
```

```
  my $client_list = $$shared_port_table{$shared_port_name};
```

```
45  # Look at all the clients. Build-up enough information
  # to calculate the port-width (below). Also, check that
  # all the clients have a consistent direction, avalon-role, etc.
  #
```

```
50  my $client_dir      = "";
  my $shared_port_role = "";
  my $shared_port_width = 0;
  foreach $client_port (@$client_list)
```

```
55  {
    $shared_port_role = $$client_port{avalon_role} if !$shared_port_role;
    $client_dir       = $$client_port{direction}  if !$client_dir;
```

```
    $client_dir eq $$client_port{direction} or die "
      Shared port $shared_port_name has client $$client_port{name}
      (direction is $$client_port{direction}; expected $client_dir).";
```

```
60    $shared_port_role eq $$client_port{avalon_role} or die "
      Shared port $shared_port_name (role: $shared_port_role) has
```

09880106-061201
102190-90108860

client \$\$client_port(name) (role: \$\$client_port(avalon_role)).";

#####

Computing the width is tricky, but only for address ports.
For everybody else, it's just the width of the widest client.
For address ports, we have to take into account the fact that
-all- shared address ports present the full byte-address,
all the way down to A[0] (even if none of the clients use
A[0]). Thus, the width of the shared port might be greater
than the width of any (all) of the client address ports.

if (\$shared_port_role eq "address")
{
Address port.. Now we must inquire about its ancestry:
my \$parent_module = \$\$client_port(parent);
my \$net_width = \$\$parent_module(highest_address_bit_used) + 1;

\$shared_port_width = \$net_width
if \$net_width > \$shared_port_width;
} else {
Normal, non-address port: Width is just the max of clients:

\$shared_port_width = \$\$client_port(width)
if \$\$client_port(width) > \$shared_port_width;

}
} # End: Client-port loop.

#####

Shared data-bus ports.

There's one weird circumstance where a shared-port is actually
wider than the max of its clients: When it's the main
tri-state data bus to the CPU. In this one case, the bus
must be at least as wide as the CPU

if ((\$Sys(Principal_Tri_State_Data_Bus)) &&
(\$shared_port_role eq "data") &&
(\$shared_port_name =~ /^\$\$Sys(Principal_Tri_State_Data_Bus)/)) {
\$shared_port_width = \$\$Sys(master_data_width);
}

Shared ports always have an avalon-role, so they always show up
on the PBM as the -complimentary- match for the client-port.
also, shared ports are always external, so they are always promoted
to like-named system-level ports.

my \$shared_port_dir = \$Complementary_Direction(\$client_dir);
my \$shared_port_description =
"\$shared_port_name | \$shared_port_width | \$shared_port_dir,";

\$\$Sys(pbm_list_ports_for_string) .= \$shared_port_description;
\$\$Sys(system_list_ports_for_string) .= \$shared_port_description;

}

We've built those nice list-ports-for string, so let's use 'em:

&List_Ports_For (\$\$Sys(name), \$\$Sys(system_list_ports_for_string));
&List_Ports_For (\$\$Sys(core_name), \$\$Sys(system_list_ports_for_string));
&List_Ports_For (\$\$Sys(pbm_name), \$\$Sys(pbm_list_ports_for_string));

#####


```

# Core_Emit_Wire_Declarations
#
# The system's "core" module is just a bunch of instances all wired-up
# together. Some "wires" in the core-module are actually external
5 # ports, so they're already declared in the cores' module
# declaration. Other "wires" are actual Verilog wires which we
# need to declare. They are the internal (module-to-module) signals
# which are never seen from the outside.
#
10 # This function emits wire-declarations for all the internal signals.
#
# It's easy to identify all the internal signals: there's one for
# every module-port scoped "internal" or "master" -- in other words,
# there's a core-level wire for every non-external port on every
15 # module in the system. That makes it pretty easy:
#
#####
sub Core_Emit_Wire_Declarations
{
20     my ($Sys) = (@_);

    foreach $Mod (&Get_Sys_Module_List($Sys)) {
        foreach $Port (&Get_Module_Port_List ($Mod))
        {
25             next if $$Port{is_external};
            my $range = &W($$Port{width});
            &Vprint ("wire $range $$Port{system_signal};\n");
        }
    }
30 }

#####
# Core_Emit_Instances
#
# The system's core-module is a bunch of instances all wired-up
# together. This function emits the instantiation-statements
# for all system-modules, including the PBM.
#
# In general, any given port on a module does not necessarily
# connect to a core-level signal (wire) of that same name.
40 # The correspondence of what-signal-goes-to-what-port, though,
# is built-in to the %Sys-database hash. We extract this
# information and use it to instantiate each module in the
# system.
45 #
#####
sub Core_Emit_Instances
{
    my ($Sys) = (@_);
50
    #####
    # Start with an instantiation of the PBM.
    #
    # Note that the PBM doesn't appear on the system's {module_table}.
    # Note also that all the ports on the PBM -do- connect to like-named
55 # signals at the core-level. This makes instantiation a snap:

    &Instantiate_And_Connect ($$Sys{pbm_name}, "the_$$Sys{pbm_name}");

60
    #####
    # Now instantiate all the "regular" modules. But, before we do,
    # go through their ports and build up a correspondence ("exception")
    # table:

```

```

#
foreach $Mod (&Get_Sys_Module_List($Sys))
{
    # Skip external modules, of course.
    next unless $$Mod{Instantiate_In_System_Module};

    my $list_ports_for_string = "";
    my %except;
    undef %except; # Superstition. I have no idea if this is called-for.

    foreach $Port (&Get_Module_Port_List ($Mod))
    {
        $except{$$Port{name}} = $$Port{system_signal};
        $list_ports_for_string .= "
            $$Port{name} | $$Port{width} | $$Port{direction},";
    }

    &List_Ports_For ($$Mod{name}, $list_ports_for_string);
    &Instantiate_And_Connect ($$Mod{name}, "the_$$Mod{name}", "", \%except);
}

#####
# PBM_Emit_Dynamic_Bus_Sizer
#
# The dynamic bus-sizer makes any narrow peripheral "look like"
# it's full-width memory. For example, if you connect an 8-bit
# memory device to a 32-bit master using DBS, the CPU will "see"
# a full 32-bit value every time it does a LD- from the device.
#
# This is accomplished, of course, through a bunch of sequential
# logic (the DBS), which (in this example) fetches four successive
# bytes from the memory, assembles them into a full-width word,
# and presents it to the CPU (all the while the CPU has been held
# in a wait-state, of course).
#
# A similar thing happens during write-operations. When
# writing a 32-bit value to 8-bit memory (for example), the CPU will
# wait while we execute four successive write-operations to the
# memory. The DBS-logic is sensitive to the fact that sometimes
# the CPU will execute a narrow-write (e.g. a byte-write). In these
# cases, the DBS-logic is smart enough to execute only one
# write-cycle.
#
# Note that all of our "successive read operations" or "successive
# write operations" are subject to the bus-timing (wait states,
# setup/hold-times, etc.) for the target peripheral. In other words,
# each sub-operation of the DBS-unit is a full-fledged bus transaction
# controlled by the wait-state generator. You could, then, think
# of the DBS-operation being "laid on top of" the regular wait-state
# logic.
#
# FYI, dynamic bus sizing is very handy for memory-type devices,
# but doesn't make much sense for register-controlled peripherals
# (e.g. UARTs).
#
# This function here creates the dynamic bus-sizing logic.
# Note that there are, really, two independent DBS-units--one
# of which handles byte-wide peripherals, the other handles
# halfword-wide peripherals.
#
# KNOWN LIMITATION:
#

```

09880105-051201

```

# If you have an 8-bit memory and you do a 16-bit write to it
# (any ST16 instruction)--you lose.
# 3 out of 3 engineers agree that DBS should handle this case.
# I'll do it when everything else works
#
#####
sub PBM_Emit_Dynamic_Bus_Sizer
{
    my ($Sys) = (@_);

    &Emit_Comment ("\n    Dynamic Bus Sizing \n");

    #####
    # First, gather-up a list of all dynamically-sized modules.
    # segregate by 8- and 16-bits wide:
    #
    my @dbs_8_modules = ();
    my @dbs_16_modules = ();

    foreach $Mod (&Get_Sys_Slave_List($Sys))
    {
        next if !$Mod{is_dynamically_sized}; # Dynamic-only, please.

        if ($Mod{Data_Width} <= 8) {
            push (@dbs_8_modules, $Mod);

        } elsif ($Mod{Data_Width} <= 16) {
            push (@dbs_16_modules, $Mod);
        }
    }

    #####
    # Or-together relevant peripherals' active signals. This
    # tells us "when to go."
    #
    # The result is a signal called "dbs_active," which
    # is true whenever the currently-selected (active) peripheral
    # uses dynamic bus-sizing.
    #
    # Note that this is -not- a state- or sequencing-bit. It
    # doesn't "say" where in the DBS-process we are. We have other
    # bits, later, which do that.
    #
    # We start the active-list off with "1'b0" as a trick to make everything
    # work out, even when the lists are empty.
    #
    my @dbs_8_active_signals = ("1'b0");
    my @dbs_16_active_signals = ("1'b0");

    foreach $Mod (@dbs_8_modules)
    {push (@dbs_8_active_signals, $Mod{internal_active_signal})}
    foreach $Mod (@dbs_16_modules)
    {push (@dbs_16_active_signals, $Mod{internal_active_signal})}

    &PBM_Assign ("dbs_8_active", join (" || ", @dbs_8_active_signals));
    &PBM_Assign ("dbs_16_active", join (" || ", @dbs_16_active_signals));
    &PBM_Wire ("dbs_active", 1, "dbs_8_active || dbs_16_active");

    #####
    # Sequencing.
    #

```

09880106 "061201

```

# We need to answer two questions:
#
# 1) When do we start a DBS-operation?
# 2) When do we "advance" to the next sub-operation (chunk)?
5 #
# Well, we start a new operation (1) when:
#
# 1a) The active peripheral needs dynamic bus-sizing.
#
10 # 1b) It's the start of a new bus-transaction
# (indicated by the "bus_transaction_start" signal,
# which is computed by the wait-state generator.
#
# 1c) There's not some other special, weird circumstance--like
15 # narrow writes or "ifetch" going on which preclude the
# need for a dynamic operation.
#
# And we advance to the next chunk (2) at the end of each
# bus cycle. Happily, the wait-state generator also indicates
20 # this by computing the signal "bus_cycle_end".
#
#
&PBM_Wire ("bus_cycle_end"); # Declared here, computed later.
&PBM_Wire ("bus_transaction_start"); # Declared here, computed later.
25
my $be_bus = &Get_Sys_Signal($$Sys{master}, "byteenablen");
&PBM_Wire ("write_is_narrow", 1,
           "| " . $be_bus);
30
if ($$Sys{master_data_width} == 32) {
    &PBM_Wire ("write_is_narrow_16", 1,
              "write_is_narrow && (($be_bus\[3\] == $be_bus\[2\]) &&
                ($be_bus\[1\] == $be_bus\[0\]) )");
35
    &PBM_Wire ("dbs_8_half_start", 1, "
              dbs_8_active          &&
              bus_transaction_start &&
              (write_is_narrow_16)  ");
40
}
&PBM_Wire ("dbs_8_full_start", 1, "
              dbs_8_active          &&
              bus_transaction_start &&
              (~write_is_narrow)    ");
45
my $ifetch_signal = &Get_Sys_Signal ($$Sys{master}, ifetch);
&PBM_Wire ("dbs_16_start", 1, "
              dbs_16_active          &&
              bus_transaction_start &&
50              (~$ifetch_signal)    &&
              (~write_is_narrow)    ");

#####
# Sequencing registers
55 #
# Here are the signals we must come up with:
#
#     dbs_8_byte_{0,1,2,3}
#     dbs_16_halfword_{0,1}
60 #
#     -- and --
#
#     dbs_8_write_byte_{0,1,2,3} ("0" not actually produced)

```

09880106.061201

```

#   dbs_16_write_halfword_{0,1} ("0" not actually produced)
#   (I think, in the absence of narrow writes, these are just
#   equivalent to the non-write versions).
5   #   dbs_wait_asserted
#
my @dbs_wait_asserted_terms = ("1'b0"); # "1'b0" fixes empty list.

10  if (scalar(@dbs_8_modules))
    (
      # How many additional bus cycles? 3 or 1, for 32- and 16-bit masters.
      # These cases are similar-enough that it's tempting to make
      # one piece of code that builds both, but I'm just going to break
      # them into separate cases for clarity:
15  #
      if ($$Sys{master_data_width} == 32)
      {
        &PBM_Wire ("dbs_8_byte_3");
        &PBM_Wire ("dbs_8_byte_1");

20      &Delay ("out          = dbs_8_state_byte_3,
                sync_set    = dbs_8_full_start,
                sync_reset  = bus_cycle_end,
                reset       = ,
25              ");

        &Delay ("out          = dbs_8_byte_2,
                in           = dbs_8_state_byte_3,
                enable       = bus_cycle_end,
                reset        = ,
30              ");

        # Two versions of byte-1 signal, because it can be set from
        # two different sources: The continuation of a 4-byte
        # operation, or the start of a two-byte operation.
35      #
        &Delay ("out          = dbs_8_continue_byte_1,
                in           = dbs_8_byte_2,
                enable       = bus_cycle_end,
                reset        = ,
40              ");

        &Delay ("out          = dbs_8_state_byte_1,
                sync_set    = dbs_8_half_start,
                sync_reset  = bus_cycle_end,
                reset       = ,
45              ");

        &Delay ("out          = dbs_8_byte_0,
                in           = dbs_8_continue_byte_1 || dbs_8_state_byte_1,
                enable       = bus_cycle_end,
                reset        = ,
50              ");

        &PBM_Assign ("dbs_8_byte_3", "dbs_8_full_start      ||
                                     dbs_8_state_byte_3      ");
        &PBM_Assign ("dbs_8_byte_1", "dbs_8_half_start      ||
                                     dbs_8_state_byte_1      ||
                                     dbs_8_continue_byte_1   ");
55
        &PBM_Wire ("dbs_8_wait_asserted", 1,
                  "dbs_8_byte_3 || dbs_8_byte_2 || dbs_8_byte_1");
60

```

0930106 061201
T02T90 30T03360

```

} else {
    # 16-bit master

    &PBM_Wire ("dbs_8_byte_1");

5    &Delay ("out          = dbs_8_state_byte_1,
           sync_set      = dbs_8_full_start,
           sync_reset    = bus_cycle_end,
           reset         = ,
10          ");

    &Delay ("out          = dbs_8_byte_0,
           in            = dbs_8_state_byte_1,
           enable        = bus_cycle_end,
           reset         = ,
15          ");

    &PBM_Assign ("dbs_8_byte_1", "dbs_8_full_start || dbs_8_state_byte_1");

20    # Pedantic renaming for documentary purposes. I hope you're happy.
    #
    &PBM_Wire ("dbs_8_wait_asserted", 1, "dbs_8_byte_1");
}

25    push (@dbs_wait_asserted_terms, "dbs_8_wait_asserted");
} # End: registers for dbs_8_modules

if (scalar(@dbs_16_modules))
{
30    &PBM_Wire ("dbs_16_halfword_1");

    &Delay ("out          = dbs_16_state_halfword_1,
           sync_set      = dbs_16_start,
           sync_reset    = bus_cycle_end,
           reset         = ,
35          ");

    &Delay ("out          = dbs_16_halfword_0,
           in            = dbs_16_state_halfword_1,
           enable        = bus_cycle_end,
           reset         = ,
40          ");

    &PBM_Assign ("dbs_16_halfword_1",
45                "dbs_16_start || dbs_16_state_halfword_1");

    # Pedantic renaming for documentary purposes. I hope you're happy.
    #
    &PBM_Wire ("dbs_16_wait_asserted", 1, "dbs_16_halfword_1");

50    push (@dbs_wait_asserted_terms, "dbs_16_wait_asserted");
}

&PBM_Wire ("dbs_wait_asserted", 1, join(" || ", @dbs_wait_asserted_terms));

55    #####
    # Address-output
    #
    # The DBS-unit, of course, has to compute "altered" addresses
    # for presentation to the client peripherals. We compute
    # separate altered addresses for the DBS-8/16 units.
    #
    # We also compute a "mixed" version of the address, which we
60

```

09830106-061201

```

# make available to shared busses which contain -both- dynamically-sized
# 8- and 16-bit peripherals (I expect most systems will have no such
# bus, but we compute the address for it here, anyhow).
#
5 # Of course, these addresses only differ in the low bit(s)
# from the original address.
#
#
# Since the rules which apply to byte-enable distribution are
10 # similar (though simpler), we do "adjusted" be_n-computation
# at the same time.
#
my $sys_addr_signal = &Get_Sys_Signal ($$Sys{master}, "address");
my $sys_be_signal    = &Get_Sys_Signal ($$Sys{master}, "byteenable");
15
if (scalar(@dbs_8_modules))
{
    # We alter the low 1 or 2 address bits, depending on whether this
    # is a 16- or 32-bit system:
    #
    if ($$Sys{master_data_width} == 16)
    {
        $address_lsbs_width = 1;
        &Mux ("dbs_8_address_lsbs, type=unary, declare=1, w=1,
25         dbs_8_byte_1 --> 1'b1,
         dbs_8_byte_0 --> 1'b0,
         --> $sys_addr_signal\[0],
         ");
    } else { # 32-bit system
        $address_lsbs_width = 2;
        &Mux ("dbs_8_a0, type=unary, declare=1, w=1,
35         dbs_8_byte_3 --> 1'b1,
         dbs_8_byte_2 --> 1'b0,
         dbs_8_byte_1 --> 1'b1,
         dbs_8_byte_0 --> 1'b0,
         --> $sys_addr_signal\[0],
         ");
    }

    # For 32-bit systems, we want to leave bit 1 of the address
    # alone whenever this is a "narrow_16" access.
    &Mux ("dbs_8_a1_raw, type=unary, declare=1, w=1,
40         dbs_8_byte_3 --> 1'b1,
         dbs_8_byte_2 --> 1'b1,
         dbs_8_byte_1 --> 1'b0,
         dbs_8_byte_0 --> 1'b0,
         --> $sys_addr_signal\[1],
         ");

    &Mux ("dbs_8_a1, type=unary, declare=1, w=1,
50         write_is_narrow_16 --> $sys_addr_signal\[1],
         --> dbs_8_a1_raw");

    &PBM_Wire ("dbs_8_address_lsbs", 2, "{dbs_8_a1, dbs_8_a0}");
55 }

my $high_range = "$$Sys{master_address_width} - 1 : $address_lsbs_width";
my $high_segment= "$sys_addr_signal \[$high_range]";
&PBM_Assign ("dbs_8_address", "{$high_segment, dbs_8_address_lsbs}");
60 }

if (scalar(@dbs_16_modules))
{

```

09880106:051201

```

# This must be a 32-bit system, and we alter address-bit 1.
# Note that, by definition, this is a halfword-aligned
# address, so A[0] is neither computed nor distributed
# with this address.
#
&Mux ("dbs_16_address_lsb, type=unary, w=1, declare=1,
      dbs_16_halfword_1 --> 1'b1,
      dbs_16_halfword_0 --> 1'b0,
      --> $sys_addr_signal\[1],
      ");

&PBM_Assign ("dbs_16_address", "{
      $sys_addr_signal \[$$Sys{master_address_width}-1 : 2],
      dbs_16_address_lsb}");

# If we are accessing halfword-0, then present be[1:0] to the
# peripheral. If we are accessing halfword-1, then we present
# be[3:2] instead.
#
&Mux ("dbs_16_be_n_lsbs, type=unary, w=2, declare=1,
      dbs_16_halfword_1 --> $sys_be_signal\[3:2],
      --> $sys_be_signal\[1:0],
      ");

&PBM_Assign ("dbs_16_be_n", "{
      $sys_be_signal \[3:2],
      dbs_16_be_n_lsbs}");
}

# Create a version of "adjusted" address suitable for any occasion,
# no matter what dynamic operation is (or is not) in process. This
# address value gets distributed on shared address ports. Also
# create an "adjusted" version of the byte-enable signals.
#
# Note that, at least, this mux does collapse to something simpler
# (trivial) if we don't have any dynamic peripherals, or if one class
# (8/16) is entirely missing. That's thanks to the "1'b0" trick, above.
#
&Mux ("dbs_adjusted_address,
      type = unary,
      w     = $$Sys{master_address_width},
      dbs_8_active -->   dbs_8_address,
      dbs_16_active --> ({dbs_16_address, 1'b0}),
      --> $sys_addr_signal,
      ");

&Mux ("dbs_adjusted_be_n,
      type = unary,
      w     = 4
      dbs_16_active --> dbs_16_be_n,
      --> $sys_be_signal,
      ");

#####
# Read-data -- registers and assembled-output.
#
&PBM_Wire ("bus_cycle_data_ready"); # declared here, computed later.

if (scalar(@dbs_16_modules))
{
    &Delay ("out      = held_halfword_1,
           in        = dbs_16_muxed_input,
           w          = 16,

```

09830106 "061201


```

        enable = (dbs_16_halfword_1 && bus_cycle_data_ready),
        reset = ,
    ");
    &PBM_Assign ("dbs_16_output", "{held_halfword_1, dbs_16_muxed_input}");
5    }

    if (scalar(@dbs_8_modules))
    {
10        &Delay ("out      = held_byte_3,
                in       = dbs_8_muxed_input,
                w        = 8,
                enable = (dbs_8_byte_3 && bus_cycle_data_ready),
                reset   = ,
                ")
15        if ($$Sys{master_data_width} == 32);

        &Delay ("out      = held_byte_2,
                in       = dbs_8_muxed_input,
                w        = 8,
20        enable = (dbs_8_byte_2 && bus_cycle_data_ready),
                reset   = ,
                ")
        if ($$Sys{master_data_width} == 32);

25        &Delay ("out      = held_byte_1,
                in       = dbs_8_muxed_input,
                w        = 8,
                enable = (dbs_8_byte_1 && bus_cycle_data_ready),
                reset   = ,
30        ");

        if ($$Sys{master_data_width} == 32)
        {
            &PBM_Assign ("dbs_8_output",
                        "{held_byte_3, held_byte_2, held_byte_1, dbs_8_muxed_input}");
35        } else {
            &PBM_Assign ("dbs_8_output",
                        "{
                                held_byte_1, dbs_8_muxed_input}");
40        }
    } # End: if (scalar(@dbs_8_modules))

    #####
    # Write-data
    #
45    # Selection-mux to pick the correct segment to send.
    # This is all pretty easy, once you've got the write-control
    # signals (dbs_8_byte_3, etc) all figured-out.
    #
    # For now, the read- and write- phase control signals (e.g.
50    # dbs_8_byte_3 and friends) are the same. In the future, it might
    # be beneficial to remove the "narrow-write" exclusion-term from the
    # read-phase-control signals, but leave it in the
    # write-phase-control signals. For today, narrow writing is a
    # term in both phase-control signals--BECAUSE THEY'RE THE SAME:
55    #
    my $sys_write_data = &Get_Sys_Signal ($$Sys{master}, "writedata");

    if (scalar(@dbs_16_modules))
    {
60        &PBM_Wire ("dbs_16_write_halfword_1", 1, "dbs_16_halfword_1");

        &Mux ("dbs_16_write_data,    type = unary, w = 16,
            dbs_16_write_halfword_1 --> ($sys_write_data\{31:16}),

```

```
--> ($sys_write_data\[15:0]),
```

```

    ");
}

5  if (scalar(@dbs_8_modules))
    {
        if (($Sys{master_data_width} == 32)) {
            &PBM_Wire ("dbs_8_write_byte_3",      1, "dbs_8_byte_3");
            &PBM_Wire ("dbs_8_write_byte_2",      1, "dbs_8_byte_2");
10         }
            &PBM_Wire ("dbs_8_write_byte_1",      1, "dbs_8_byte_1");

            my $mux_string = "";
            $mux_string .= "dbs_8_write_byte_3 --> ($sys_write_data\[31:24]),
15             dbs_8_write_byte_2 --> ($sys_write_data\[23:16]),"
                if $Sys{master_data_width} == 32;
            $mux_string .= "dbs_8_write_byte_1 --> ($sys_write_data\[15: 8]),
                --> ($sys_write_data\[ 7: 0]),";

20         &Mux ("dbs_8_write_data,    type = unary, w = 8, $mux_string");
    }

    # Create a verison of "adjusted" write-data suitable for any occasion,
    # no matter what dynamic operation is (or is not) in process. This
    # write-data value gets distributed on shared data busses.
    #
    # Note that, at least, this mux does collapse to something simpler
    # (trivial) if we don't have any dynamic peripherals, or if one class
    # (8/16) is entirely missing. That's thanks to the "1'b0" trick, above.
    #
30     &Mux ("dbs_adjusted_write_data,
        type = unary,
        w      = $Sys{master_data_width},
        dbs_8_active -->  dbs_8_write_data,
35         dbs_16_active --> dbs_16_write_data,
            --> $sys_write_data,

        ");
}

40 #####
# PBM_Emit_IRQ_Prioritizer
#
# Well, compared to the dynamic bus-sizer, this is sure a piece
# of cake.
45 #
# Given a %Sys-hash (reference) , we have to emit the requisite
# IRQ prioritization logic into the currently-open file (which, we
# presume, is the PBM verilog file).
#
50 # We are responsible for driving two signals:
#
#     1) The masters' "irq"-type input.
#     2) The msters' "irqnumber"-type input.
#
55 # (1) is just a straight logical-or of all the periphs' IRQ-outputs.
# (2) is just an ordered priority-mux.
#
#####
sub PBM_Emit_IRQ_Prioritizer
60 {
    my ($Sys) = (@_);
    &Emit_Comment ("\n    IRQ Prioritizer  \n");
}

```

```

my @irq_signals = ("1'b0"); # Trick makes it work when list is empty.
my %irq_hash; # Keeps track of irq signals by number, so we can sort.

```

```

foreach $Mod (&Get_Sys_Slave_List($Sys))

```

```

{
    next if !$Mod{Has_IRQ}; # that was a short trip.

```

```

    my $irq_signal = &Get_Sys_Signal($Mod, "irq");
    $irq_hash{$Mod{IRQ_Number}} = $irq_signal;
    push (@irq_signals, $irq_signal);

```

```

}
&PBM_Assign (&Get_Sys_Signal ($Sys{master}, "irq") ,
    join (" || ", @irq_signals) );

```

```

# NOTE: We do not use the generator-library &Mux-function
# because it doesn't retain the order of the terms. That's just
# the way it works. That's OK: It's easy enough to just build
# our own question-mark-colon expression:

```

```

my $mux_expression_string = "";
foreach $irq_num (sort numerically keys(%irq_hash))
{
    $mux_expression_string .= "$irq_hash{$irq_num} ? 6'd$irq_num : ";
}

```

```

$mux_expression_string .= " 6'd63"; # Default: lowest priority.

```

```

&PBM_Assign (&Get_Sys_Signal ($Sys{master}, "irqnumber"),
    $mux_expression_string);

```

```

# Mien Got, that was easy.
}

```

```

#####
# PBM_Emit_Simple_Assignments
#

```

```

# At the top of the PBM module, there are a bunch of "simple"
# assignments. These are, for the most part, statements which
# "broadcast" master-outputs to various slave modules. For example,
# the master control signals are "simply" assigned to many of the
# slave modules.

```

```

# This function handles this issue role-by-role.

```

```

# Note that no special care is needed for shared busses, because
# we've defined the utility function &PBM_Assign so that you can
# call it multiple times to assign the same thing to the same
# signal, and it does something smart (ignores redundant assignments).
# Thus, a shared byteenable signal, for example, will get the masters'
# byteenable-output assigned to it multiple times, but it won't hurt
# anything.

```

```

# We -don't- connect writedata- or address-type ports for dynamically-sized
# peripherals. Those are generated in the DBS-unit. For the same reason,
# we also don't assign address- or writedata- signals to shared busses,
# because one of the clients might require dynamic sizing.
#

```

```

#####

```

```

sub PBM_Emit_Simple_Assignments

```

```

{
    my ($Sys) = (@_);

```

05880106 "061201

```

# Lots of things connect to the master, so let's keep a ref to the
# master's data, and its avalon-ports, handy:
my $Master_Mod = $$Sys{master};
my $master_avalon_table = $$Master_Mod{avalon_port_table};

5
&Emit_Comment("\nSimple assignments.
  Connect-up signals which pass unmolested from one PBM-port to another.");

10
&PBM_Wire ("reset", 1, "~reset_n"); # Handy: Logic-true reset.

#####
# Start with the very-easiest avalon roles:
#   always0, always1, clk, and resetn
#
15
# Even these are a little abnormal, because we have to deal with the
# fact that there could be more than one of each type.
#
foreach $Mod (&Get_Sys_Module_List($Sys)) {
  foreach $Port (&Get_Module_Port_List ($Mod))
20
  {
    if ($$Port{avalon_role} eq "always0") {
      &PBM_Assign ($$Port{system_signal}, "${$Port{width}}'b0");
    } elsif ($$Port{avalon_role} eq "always1") {
      &PBM_Assign ($$Port{system_signal}, "${$Port{width}}'b1");
25
    } elsif ($$Port{avalon_role} eq "clk") {
      &PBM_Assign ($$Port{system_signal}, "clk");
    } elsif ($$Port{avalon_role} eq "resetn") {
      &PBM_Assign ($$Port{system_signal}, "reset_n");
30
    }
  }
}

#####
# Address-broadcast.
35
#
# Each peripheral with an address-port gets the appropriate
# flavor of address -- except, of course, dynamically-sized peripherals,
# which get a special address generated by the DBS-unit.
#
40
# The byte-enable control signals could, I suppose, be considered "part
# of" the address, so they get assigned in this same loop. Once
# again, byte-enables for dynamic devices are handled elsewhere.
#
# !!!SUBOPTIMALITY NOTE!!!
45
#
# Shared-busses are complicated, because it's a bit of an ordeal to
# figure-out exactly -which- flavor of dynamic address they get--it
# depends upon what's connected to them. If we were to do this in
# some clever, optimal way, it would be a big hassle (trust me: a
50
# great big hassle).
#
# This same rule applies to the byte-enable signals.
#
# Instead, we always assign a dbs-adjusted
55
# version of the address (which comes from logic) to shared address
# ports. Even shared address ports that don't have any dynamically-sized
# clients on them. Thus, we sometimes introduce more logic-delay than
# strictly necessary. Sue me. I'll fix it if it ever becomes a
# problem.
60
#
my $byte_range_select = "$$Sys{master_address_width}-1 : 0";
my $halfword_range_select = "$$Sys{master_address_width}-1 : 1";

```

09830105 "061201

```

my      $word_range_select = "$$Sys{master_address_width}-1 :.2";
&Emit_Comment ("\\n  Address-assignments\\n");

```

```

# Emit wire-declarations for DBS-generated addresses, whether we
# need them or not.  These get assigned-to when we build the
# dynamic bus-sizers, but we (might) use them here:
#

```

```

&PBM_Wire ("dbs_16_address",      $$Sys{master_address_width}-1);
&PBM_Wire ("dbs_8_address",       $$Sys{master_address_width} );
&PBM_Wire ("dbs_adjusted_address", $$Sys{master_address_width} );
&PBM_Wire ("dbs_16_be_n",         4);
&PBM_Wire ("dbs_adjusted_be_n",   4);

```

```

foreach $Mod (&Get_Sys_Slave_List($Sys))

```

```

{
  next if $$Mod{Is_Bus_Master}; # Slaves only, please

```

```

  # Go ahead and hook-up the byte-enables.  That's straightforward.
  #

```

```

  # Note: I may need to change this later to work with narrow writes
  # through the DBS-unit.
  #

```

```

  my $BE_Port = &Get_Port_By_Role ($Mod, "byteenablen");

```

```

  if ($BE_Port) {
    if ($$BE_Port{is_shared} ||
        $$Mod{is_dynamically_sized} )

```

```

    {
      # All shared byte-enables are dbs-muxed.  How suboptimal.
      &PBM_Assign (&Get_Sys_Signal ($Mod, "byteenablen"),
                  "dbs_adjusted_be_n");

```

```

    } else { # Non-dynamic byte-enables
      &PBM_Assign (&Get_Sys_Signal ($Mod, "byteenablen"),
                  &Get_Sys_Signal ($Master_Mod, "byteenablen"));

```

```

    } # if ($BE_Port)

```

```

  my $source_signal = "";
  my $A_Port = &Get_Port_By_Role ($Mod, "address");

```

```

  if ($A_Port) {
    if ($$A_Port{is_shared})
    {
      # All shared addresses are dbs-muxed.  How suboptimal.
      $source_signal = "dbs_adjusted_address";

```

```

    }
    elsif ($$Mod{is_dynamically_sized})

```

```

    {
      $source_signal =
        $$Mod{address_type_used} eq "byte" ? "dbs_8_address" :
        "dbs_16_address";

```

```

    } else { # Non-dynamic address

```

```

      my $range_select =
        $$Mod{address_type_used} eq "byte" ? $byte_range_select :
        $$Mod{address_type_used} eq "halfword" ? $halfword_range_select :
        $word_range_select;

```

```

      $source_signal =
        &Get_Sys_Signal ($Master_Mod, "address") . "[ $range_select ]";

```

```

    }
    &PBM_Assign (&Get_Sys_Signal ($Mod, "address"), $source_signal);

```

```

  } # if ($A_Port)
} # foreach $Mod ...

```

```

#####

```

00000106 061201

```

# Data-broadcast
#
# Pretty simple: Each peripheral gets a copy of the master's write-data,
# unless it's dynamic--then it gets a special "tweaked" version of the
# data.
#
# Shared data-ports could, in theory, have any kind of peripheral attached
# to them, so they need to drive-out a fully "dynamically-adjusted"
# version of the data suitable for the current occasion. Happily, the
# DBS-unit computes this very thing for us.
#
# See the !!!SUBOPTIMALITY NOTE!!! above for addresses.
# The same thing goes here. We're penalizing the write-data path on
# busses that may not have any dynamic clients at all. For now:
# tough beans. For later, maybe we'll get more clever.
#
# Another slight suboptimality: The way this works, we drive data out
# on -all- tri-state busses whenever the master does a write. This
# might involve a slight waste of power, because we'll be wiggling I/O
# pins more than strictly necessary. Again, tough beans.
#
&Emit_Comment ("\n Data-assignments\n");

# Emit wire-declarations for DBS-generated write-data, whether we
# need them or not. These get assigned-to when we build the
# dynamic bus-sizers, but we (might) use them here:
#
&PBM_Wire ("dbs_16_write_data", 16);
&PBM_Wire ("dbs_8_write_data", 8);
&PBM_Wire ("dbs_adjusted_write_data", $$Sys(master_data_width));

foreach $Mod (&Get_Sys_Slave_List($Sys))
{
    my $source_signal = "";
    my $target_signal = &Get_Sys_Signal ($Mod, "writedata");

    if ($$Mod{Uses_Tri_State_Data_Bus})
    {
        # The "principal" tri-state bus is managed as part of the
        # read-data path (elsewhere).
        next if $$Mod {Tri_State_Data_Bus} eq
            $$Sys(Principal_Tri_State_Data_Bus) ;

        # For tri-state busses, we need to use the "data"-type port
        # instead of the "writedata"-type port as the assignment
        # target:
        #
        $target_signal = &Get_Sys_Signal ($Mod, "data");

        # To this bus we will assign a value which, as it
        # happens, incorporates its whole tri-state logic. Perversely,
        # we may "PBM_Assign" this same expression to this bus several
        # times, but it doesn't hurt anything.
        #
        my $sys_writen = &Get_Sys_Signal ($$Sys(master), "writen");
        $source_signal = "(~$sys_writen) ? dbs_adjusted_write_data :
            $$Sys(master_data_width)'bZ ";

    }
    elseif ($$Mod{is_dynamically_sized})
    {
        $source_signal =
            $$Mod(address_type_used) eq "byte" ? "dbs_8_write_data" :
            "dbs_16_write_data" ;
    }
}

```

09830106 "061201

```

    } else { # Non-dynamic address
        $source_signal = &Get_Sys_Signal ($Master_Mod, "writedata");
    }

5    &PBM_Assign ($target_signal, $source_signal);
}

#####
# Readn/Writen -broadcast.
10 #
# Every module gets a copy of the master's "readn" and "writen" signals
# -- unless they have a nonzero setup- or hold-time, in which case
# they get their very-own custom-made, tweaked copy of these signals.
#
15 # That all happens later when the wait-state unit gets created.
#
&Emit_Comment ("\n    Control-assignments\n");
foreach $Mod (&Get_Sys_Slave_List($Sys))
{
20     next if $$Mod{Setup_Time} != 0;
    next if $$Mod{hold_time_full_clocks} != 0;

    &PBM_Assign (&Get_Sys_Signal ($Mod, "readn"),
                 &Get_Sys_Signal ($Master_Mod, "readn"));

25     &PBM_Assign (&Get_Sys_Signal ($Mod, "writen"),
                 &Get_Sys_Signal ($Master_Mod, "writen"))
        unless $$Mod{Hold_Time}; # half-cycle hold needs custom write strobe.
}
30 }

#####
# PBM_Emit_Address_Decoder
#
35 # Given a ref to the %Sys-hash, we can generate all the logic
# to build the address decoder.
#
# This function emits all the Verilog statements which
# implement the address-decoder directly into the currently-selected
40 # output file. These statements include:
#
# * wire-declarations for modules that don't have chip-select ports.
# * Logic-assignments to all "active-" signals.
# * Instantiate fast I/O registers for "registeredselectn"s, if any.
45 #
#####

sub PBM_Emit_Address_Decoder
{
50     my ($Sys) = (@_);

    &Emit_Comment ("\n    Address decoder \n");

    # Extract the name of the masters' address signal:
55     my $Master_Mod = $$Sys{master};
    my $master_address = &Get_Sys_Signal($Master_Mod, "address");

    # external chip selects may get gated with slave_phase below.
    &Vprint("reg slave_phase;\n");
60     foreach $Mod (&Get_Sys_Slave_List($Sys))
    {
        $$Mod{external_active_signal} = &Get_Sys_Signal ($Mod, "chipselect");
    }
}

```

```

# All modules get a pbm_internal select signal regardless of if they
# need an external chip select signal. This is because
# we use address-decode signals to operate the
5 # read-data mux and wait state timer. PBM_external signals have
# gotten a tad more tricky. We now gate them with slave_phase.
# This may
#
# All modules, including ones requiring the evil
10 # "registeredselectn"-type signals also fall into this category.

$$Mod(internal_active_signal) = "$$Mod(name)_active";

# How quaint that we still call this old warhorse of a function:
15 #
&Make_Nios_Chip_Select ("out          = $$Mod(internal_active_signal),
                        device_base   = $$Mod(Base_Address),
                        device_span   = $$Mod(address_span),
                        outer_span    = $$Sys(address_span),
20 address_name = $master_address,
                        ");

if ($$Mod(external_active_signal))
{
25   if (
        ($$Mod(Read_Wait_States) eq "peripheral_controlled") ||
        ($$Mod(Write_Wait_States) eq "peripheral_controlled")
    )
    {
30       &PBM_Wire ($$Mod(external_active_signal),1,
                  "$$Mod(internal_active_signal) & slave_phase");
    }
    else
    {
35       &PBM_Wire ($$Mod(external_active_signal),1,
                  $$Mod(internal_active_signal));
    }
}
40

#####
# Registered chip-selects.
#
# The whole dirty little business of registered chip-select signals
45 # is, I suppose, part of the address-decoding job. Note
# that modules can have more than one registered chip-select signal,
# each of which is derived (of course) from that modules' active-signal.
#
# There's also a wait-state aspect to this task (wait-states
50 # occur as a consequence of the chip-select delays). That's not
# handled here--it's handled later, when we build the wait-state
# unit.
#
my $Fast_IO_Setting      = "OFF";
55 my $chipselect_reg_module = $$Sys(name) . "_rg";

if ($$Sys(has_registered_select_signals))
{
    $Fast_IO_Setting = "ON";
60
    # First, we create a special flip-flop module just for the
    # purpose. It needs to be "firm" so that synthesis tools don't
    # optimize-away duplicates, and so that we can make a

```

09330106-061201


```

# FAST_OUTPUT_REGISTER entity-assignment to it without "losing" its
# name.
#
5  &Create_Firm_Flip_Flop_Variant ($$Sys(system_directory),
                                $$Sys(sopc_directory),
                                $$Sys(device_family),
                                $chipselect_reg_module,
                                );

10  # Push this register onto the list of files to be synthesized.
    my $synth_list_ref = $$Sys(synth_file_list);
    push (@$synth_list_ref,
          "$$Sys(system_directory)/$chipselect_reg_module.v");

15  #&Emit_Comment
      (q[
        Registered chip-select signals

20  A module may optionally request one (or more) -registered- chip-select
    signals (by declaring a port of type "registeredselectn", and by
    setting the "Uses_Registered_Select_Signal" assignment TRUE in the
    SYSTEM_BUILDER_INFO section).

25  Modules do this when they have stringent setup-time requirements
    on their select-signals. Such stringent setup requirements are
    greatly assisted by having the select-signal be produced from
    an Apex Fast-I/O register, right in the pad structure itself.

30  Delaying (registering) the select-signal to a module is tricky
    business, but the PBM "absorbs" all the trickiness and does the right
    thing. Wait-states get generated (in subsequent code) when the
    (delayed) select-signal being fed to the device disagrees with its
    correct, current value. The logic is clever enough to allow successive
    accesses to this same device with no wait-penalties. This
35  arrangement is eminently suitable for external asynchronous RAM
    used as main-memory.

    Registered chip-selets are always logic-negative, because that's the
    way chip-level select signals have been since the dawn of time.
40  Because these come directly from fast I/O registers, there is no
    subsequent opportunity to negate them, or even copy them to other pins.

    We accomplish all this using deep voodoo magic: We use a special
    "Firm Flip-Flop" module, since this is the only way to convince
45  the synthesis tool to -not- optimize-out "redundant" chip-select
    registers. It also provides a handy method for assigning the
    'FAST_OUTPUT_REGISTER' attribute--using an ESF-file.
    ]);

50  foreach $Mod (&Get_Sys_Slave_List($Sys)) {
    foreach $Port (&Get_Module_Port_List ($Mod))
    {
        next if $$Port(avalon_role) ne "registeredselectn";

55        # If we got to here, then we know %$Port is
        # of type "registeredselectn". Blurt-out an instance
        # of a properly-connected firm flip-flop with no further ado:
        #
        my $instance_name = "$$Mod{name}_$$Port{name}_delay_register";
        &Vprint ("
60        $chipselect_reg_module $instance_name
            (
                .clk      (clk),

```

09330106-061201
 09330106-061201
 09330106-061201

```

        .ena      (1'b1),
        .clrn     (reset_n),
        .prn      (1'b1),
        .d         (~$$Mod{internal_active_signal}),
        .q         ($$Port{system_signal})
    );
    // exemplar attribute $instance_name NOOT TRUE
    ");
}

} # End: if ($$Sys{has_registered_select_signals})

# We -always- create an ESF file because:
# 1) don't cost nuthin'
# 2) Un-sets FAST-I/O req'mnt if it had been previously set.
#
&Create_ESF_File
    ("firm_flip_flop : FAST_OUTPUT_REGISTER = $Fast_IO_Setting;",
     $$Sys(system_directory),
     $chipselect_reg_module);

}

#####
# PBM_Emit_Read_Data_Mux
#
# Given a ref to the %Sys-hash, we can generate all the logic
# to build the read-data mux path.
#
# This function emits all the Verilog statements which
# implement the read-data mux structure directly into the
# currently-selected output file.
#
# **** Mux structure
#
# We build-up a "fast mux" and a "slow mux." The "fast mux"
# gathers-up all the readdata-type signals for zero-wait-state
# modules (if any). The slow-mux gathers-up all the others.
#
# **** The "principal" bus.
#
# The road to the CPU takes another twist if you have a "principal"
# tri-state data bus. The principal data bus gets routed -directly-
# to the CPU's read-data port--the rest of the (muxed-together) signals
# a route "around the horn": Driven-out onto the principal
# databus in order to be read back in.
#
# If the CPU doesn't have any such "principal" tri-state busses, then
# things are simpler--the muxes just collect the data.
#
# **** Widths

# In this age of dynamic bus-sizing, all narrow data busses are just
# zero-padded. This happens "almost invisibly" in Verilog syntax when
# we assign a narrow value to a wide target.
#
# If a peripheral is "dynamic," then we use the ouptut
# from the appropriate DBS-unit in place of its data. The DBS-unit
# itself is generated later. And, to help-out the DBS-unit, we
# also generate its input-muxes right here, because we are already
# engaged in the process of building mux-like things.
#

```

09330105 "061201

```

#####
sub PBM_Emit_Read_Data_Mux
{
5   my ($Sys) = (@_);
   &Emit_Comment ("\n    Read-Data Multiplexer \n");

   #####
   # Loop to build-up slow- and fast-mux strings.
   #
10  # Also, build-up mux strings for the dynamic bus-sizers' inputs.
   #
   my $fast_mux_string = "";
   my $slow_mux_string = "";
15  my $dbs_8_mux_string = "";
   my $dbs_16_mux_string = "";

   foreach $Mod (&Get_Sys_Slave_List($Sys))
   {
20     my $data_signal = &Get_Sys_Signal ($Mod, "readdata");
       $data_signal = &Get_Sys_Signal ($Mod, "data")
           if $$Mod{Uses_Tri_State_Data_Bus};

       next if $data_signal eq ""; # Some peripherals are write-only (!)

25     if ($$Mod{is_dynamically_sized})
       {
           # Yikes. A dynamically-sized peripheral. Its data comes from
           # the appropriately-sized DBS-unit. Note that dynamically-sized
           # peripherals are automatically assumed to be "slow."
           #
30         if ($$Mod{Data_Width} <= 8)
           {
               $slow_mux_string      .= "$$Mod{internal_active_signal}  -->
dbs_8_output, ";
35         $dbs_8_mux_string      .= "$$Mod{internal_active_signal}  -->
$dbs_8_mux_string, ";
           } else {
               $slow_mux_string      .= "$$Mod{internal_active_signal}  -->
dbs_16_output, ";
40         $dbs_16_mux_string     .= "$$Mod{internal_active_signal}  -->
$dbs_16_mux_string, ";
           }

           } else {
45         # Not dynamic--it's ether a slow-thing or a fast-thing.
           #
           # Principal data-bus is not (directly) part of this mux.
           # but we are accutely interested in the active-signals
           # for peripherals which sit on the principal bus. We'll
50         # use these signals later to control the bus-direction.
           next if $$Mod{Uses_Tri_State_Data_Bus} &&
               ($$Mod{Tri_State_Data_Bus} eq $$Sys{Principal_Tri_State_Data_Bus});

           if ($$Mod{Read_Wait_States} == 0) {
55             $fast_mux_string      .= "$$Mod{internal_active_signal}  -->
$dbs_8_mux_string, ";
           } else {
               $slow_mux_string      .= "$$Mod{internal_active_signal}  -->
60             $dbs_16_mux_string, ";
           }
       }
   } # End: $Mod-loop.
}

```

09880106 "061201

```

# There may or may not be input-muxes to the DBS-units:
#   if so, we declare the DBS-unit's output wire here
#   (becuase we have to). But the DBS-units themselves are made later.
#
5  if ($dbs_8_mux_string)
    {
        &Modified_Mux("dbs_8_muxed_input, type=unary, w=8, $dbs_8_mux_string");
        &PBM_Wire ("dbs_8_output", $$Sys{master_data_width});
    }
10
    if ($dbs_16_mux_string)
    {
        &Modified_Mux("dbs_16_muxed_input, type=unary, w=16,
15         $dbs_16_mux_string");
        &PBM_Wire ("dbs_16_output", $$Sys{master_data_width});
    }

# There may or may not be a slow mux:
#
20 if ($slow_mux_string)
    {
        &Modified_Mux ("slow_read_data_mux_output, type=unary,
            w = $$Sys{master_data_width}, $slow_mux_string
            ");

25         # The "default" input to the fast-mux is, of course, the output
        # of the slow-mux:
        #
        $fast_mux_string .= " --> slow_read_data_mux_output,";
30     }

# There is always a fast-mux:
&Modified_Mux ("read_data_mux_output, type=unary,
    w = $$Sys{master_data_width},
35     $fast_mux_string
    ");

#####
# Data-in to CPU.
40 #
# Well, gee. There are two cases here. Let's start with the
# easiest:
#
# **** -NO- principal tri-state bus.
45 #
# In this case, the fast-mux ouptut goes right to the
# CPU's data-input, and that's that.
#
# **** Principal tri-state bus.
50 #
# The principal tri-state bus actually drives the CPU's
# read-data input directly. The fast-mux output gets registered.
# The output of this register gets driven-out onto the principal
# bus (and, therefore, to the CPU). Thus, perversely, we -drive-
55 # data out onto this bus at the end of any bus read-transaction
# which did not get data from the principal bus (and, even then,
# we do it if there's dynamic bus-sizing).
#
&PBM_Wire ("data_driveback_active"); # These Declare here, compute later.
60 &PBM_Wire ("data_driveback_asserted");
&PBM_Wire ("dbs_8_active");
&PBM_Wire ("dbs_16_active");

```

03880106 061201
T02130" 90108865

```

if (!$$Sys(Principal_Tri_State_Data_Bus))
{
    # The easy case: Mux feeds CPU, period.
    #
5    &PBM_Assign (&Get_Sys_Signal ($$Sys(master), "readdata"),
                "read_data_mux_output");
} else {
    # Ugh.

10    # To start with, hook-up principal bus directly to CPU's data-in:
    #
    &PBM_Assign (&Get_Sys_Signal ($$Sys(master), "readdata"),
                "$$Sys(Principal_Tri_State_Data_Bus)_data");

15    # Next, run the result of the read-data mux into a holding
    # register:
    &Delay("in      = read_data_mux_output,
          w        = $$Sys(master_data_width),
          reset    = ,
20          ");

    #####
    # What to drive?
    #
25    # We can drive one of two things out on the principal data
    # bus:
    #    1) Outbound write-data from the CPU.
    #        For this purpose, we use the same "dbs_adjusted_write_data"
    #        that every other tri-state bus gets. This is suboptimal
30    #        in the same way that it is for every other tri-state bus.
    #
    #    3) Data from the read-mux being sent "back out" to the CPU.
    #
    # Those are the only two possibilities, and we can tell which
35    # to use just by looking at the masters' written-signal:
    #
    my $master_write_n = &Get_Sys_Signal ($$Sys(master), "written");

    &Mux ("principal_data_drive_value, type=unary, declare=1,
40          w = $$Sys(master_data_width),
          (~$master_write_n) --> dbs_adjusted_write_data,
          --> d1_read_data_mux_output,

          ");

45    #####
    # When to drive?
    #
    # Like every other tri-state bus, we drive data during actual
50    # bona-fide write-operations. We also drive data during the
    # "driveback" phase, as-determined by a signal named:
    #
    #    data_driveback_asserted
    #
    # which (we presume) gets computed by the wait-state generator.
55    #
    # **** Interaction: Dynamic Bus Sizing
    #
    # And, of course, there's a bit of strangeness when a device
60    # on the principal databus is dynamically-sized. The easiest
    # way to think about this: Set aside dynamic-sizing for a
    # moment, and think only about narrow peripherals on the
    # principal databus (be they dynamically sized or no).

```

```

#
# Narrow peripherals, by definition, only drive low
# bits (e.g. LS-byte or LS-halfword) of the databus. So,
# when reading from (say) an 8-bit device on a tri-state databus,
5 # the high-order bits [31..8] aren't driven by anyone. There's
# no harm in -us-driving these bits, now, is there? If we
# don't, then they'll just read as <undefined> ('Z', in
# simulation), and that's not really any help to anyone.
#
10 # So let's suppose we decide to be good citizens and "nail-down"
# the high-order address bits when a narrow peripheral is
# accessed on the principal tri-state bus. This gives us a warm
# feeling inside, and, as an added bonus, makes dynamic bus
# sizing work for devices on the principal bus (the high-bits
15 # get driven from the appropriate dbs-holding registers. Yay.
#
# So, despite the heft of this comment, the actual solution is
# both simple and tidy:

20 &PBM_Wire ("do_drive_principal_data_bus_byte_0", 1,
            "~$master_write_n) || data_driveback_active");

&PBM_Wire ("do_drive_principal_data_bus_byte_1", 1,
            "do_drive_principal_data_bus_byte_0 || dbs_8_active");

25 &PBM_Wire ("do_drive_principal_data_bus_bytes_2_and_3", 1,
            "do_drive_principal_data_bus_byte_0 ||
            dbs_16_active || dbs_8_active");

30 # Assign principal databus in bitwise-chunks.
#
&PBM_Assign("$$Sys{Principal_Tri_State_Data_Bus}_data[7:0]",
            "(do_drive_principal_data_bus_byte_0 ?
              principal_data_drive_value[7:0] :
35              8'bZ
            ");

&PBM_Assign("$$Sys{Principal_Tri_State_Data_Bus}_data[15:8]",
            "(do_drive_principal_data_bus_byte_1 ?
              principal_data_drive_value[15:8] :
40              8'bZ
            ");

if ($$Sys{master_data_width} == 32) {
45   &PBM_Assign("$$Sys{Principal_Tri_State_Data_Bus}_data[31:16]",
               "(do_drive_principal_data_bus_bytes_2_and_3 ?
                 principal_data_drive_value[31:16] :
                 16'bZ
               ");
50 }
}

#####
# "memis32bits"
55 #
# Somebody needs to tell the master when it's fetching
# 32-bit data--that's just one of the inputs that an Avalon master
# might require.
#
60 # I couldn't say for sure that this really belongs in the "Read Data Mux"
# generator, but I can't think of where else to put it, so here
# it is.
#

```

```

# Note that 16-bit dynamically-sized devices appear as 32 bits
# -unless- "ifetch" is asserted, in which case they don't. 8-bit
# dynamically-sized devices always just appear as 32-bits wide.
#
5 # It might be more efficient to say when something -doesn't- return
# a 32-bit value. But For now, I do the logic-true computation:
#
my $ifetch_signal = &Get_Sys_Signal ($$Sys{master}, "ifetch");

10 my @memis32bits_terms = ("1'b0");
foreach $Mod (&Get_Sys_Slave_List ($$Sys))
{
    if ($$Mod{Data_Width} > 16) {
        push (@memis32bits_terms, $$Mod{internal_active_signal});
15     } else {
        if ($$Mod{is_dynamically_sized}) {
            if ($$Mod{Data_Width} <= 8) {
                push (@memis32bits_terms, $$Mod{internal_active_signal});
            } else {
20                 push (@memis32bits_terms,
                        "$$Mod{internal_active_signal} && (~$ifetch_signal)");
            }
        }
    }
25 }

&PBM_Assign (&Get_Sys_Signal ($$Sys{master}, "memis32bits"),
             join (" || ", @memis32bits_terms));

30 }

#####
# PBM_Emit_Wait_State_Generator
#
35 # Given a ref to the %Sys-hash, we generate all the logic
# to build the read-data mux path. The logic gets
# dumped-into the currently-selected output file.
#
#####
40 sub PBM_Emit_Wait_State_Generator
{
    my ($$Sys) = (@_);
    &Emit_Comment ("\n    Wait-State Generator \n");

45     # This function seems preternaturally-concerned with the read-
    # and write-strobes. Assign their signal-names to some
    # handy local variables:
    #
    my $sys_readn = &Get_Sys_Signal ($$Sys{master}, "readn");
50     my $sys_writen = &Get_Sys_Signal ($$Sys{master}, "writen");

    #####
    # Peripheral-Controlled Wait
    #
55     # If the currently-selected peripheral has a wait-request
    # signal, and it's asserting it, then we definitely want to
    # wait. This is pretty easy to compute:
    #
    my @periph_controlled_wait_terms = ("1'b0"); # 1'b0 fixes empty lists.

60     foreach $Mod (&Get_Sys_Slave_List($$Sys))
    {

```

T02T90"90T0886

```
my $request_signal = &Get_Sys_Signal ($Mod, "waitrequest");
```

```
#####
```

```
# @periph_controlled_wait_terms is one of the very few things  
# (only one at the moment) that gets an  
# external_active_signal. We cut off the external select  
# after the peripheral lowers its wait pin. It doesn't get to  
# change its mind later on in the same bus cycle.
```

```
push (@periph_controlled_wait_terms,  
      "$Mod(external_active_signal)    &&    $request_signal    &&  
(~$sys_writen)"  
      ) if $$Mod{Write_Wait_States} eq "peripheral_controlled";
```

```
push (@periph_controlled_wait_terms,  
      "$Mod(external_active_signal) && $request_signal && (~$sys_readn)"  
      ) if $$Mod{Read_Wait_States} eq "peripheral_controlled";
```

```
}
```

```
&PBM_Wire ("periph_wait_asserted", 1,  
           join (" || ", @periph_controlled_wait_terms));
```

```
#####
```

```
# Calculate minimum wait states.
```

```
#  
# Each module has a certain minimum number read/write wait-states,  
# perhaps even including zero. This is the sum of the  
# user-supplied wait-states and, IN ADDITION,  
# any setup- and hold-times the module may require.
```

```
#  
# It may seem a bit strange, but even modules with  
# "peripheral_controlled" wait-states can also have a  
# minimum number of fixed wait-states--that's because they may  
# also require setup/hold time.
```

```
#  
# The minimum number of wait-states we calculate here (and save  
# as part of each %Mod-hash) DOES NOT INCLUDE delays due to  
# dynamic bus sizing or principal-databus driveback.
```

```
#  
# Here we just compute the minimum number of clocks -In A Single  
# Bus Cycle-. If the peripheral requires multiple bus cycles  
# (via dynamic bus sizing) or requires an extra clock for databus  
# drive-back, that's not our problem here.
```

```
#  
# The values we compute here (minus 1) get loaded into the  
# wait-state counter when any of these peripherals are selected.
```

```
#  
# while we're looping through the values, take note of the  
# largest number we'll see--this number (less 1) will be the biggest  
# value we ever load into the wait-state counter.
```

```
#  
# **** Hold-time policy:
```

```
#  
# Giving a peripheral extra hold-time doesn't make much sense  
# following a read-cycle--though one can contrive a case where  
# you need it. We could define it either way-- hold-times apply  
# only to write-cycles, or they apply to both read- and write-cycles.  
# I've defined it so that hold-times only apply to write-cycles. It  
# makes computation of the "bus_cycle_data_ready" signal simpler.  
#
```

09330106.061201
102190" 90108860


```

# Deal with nasty case of no wait-states whatsoever, which will
# result in a zero-bit mux/counter. To deal with that case, we
# just always assume at least one wait-state:
#

```

```

5 my $max_counter_value = 1;

```

```

foreach $Mod (&Get_Sys_Slave_List($Sys))
{

```

```

10   my $min_read  = $$Mod {Read_Wait_States};
   my $min_write = $$Mod {Write_Wait_States};

```

```

   $min_read = 0 if $min_read eq "peripheral_controlled";
   $min_write = 0 if $min_write eq "peripheral_controlled";

```

```

15   $min_read += $$Mod {Setup_Time};
   $min_write += $$Mod {Setup_Time} + $$Mod {hold_time_full_clocks};

```

```

   $$Mod {read_min_wait_states} = $min_read;
   $$Mod {write_min_wait_states} = $min_write;

```

```

20   $max_counter_value =
       &Find_Max ($max_counter_value, $min_read, $min_write);
}

```

```

25 my $wait_counter_bits = &Bits_To_Encode($max_counter_value);

```

```

#####

```

```

# Counter-load mux

```

```

#

```

```

30 # Select what value goes into the wait-state counter
# based on the peripherals' active-signals (and, of course,
# whether we're reading or writing).

```

```

# Notice that we load the counter with the modules' number of
# wait-states MINUS ONE.

```

```

# Trick: All these muxes are very similar, so we use the same
# description-string with a different word at the front,
# which gives each a different output-signal:

```

```

40 my $counter_mux_string = "counter_load_value,
                           w      = $wait_counter_bits,
                           type   = unary,
                           --> 0,

```

```

45   ";

   my $write_counter_mux_string = "write_" . $counter_mux_string;
   my $read_counter_mux_string = "read_" . $counter_mux_string;

```

```

50   foreach $Mod (&Get_Sys_Slave_List($Sys))
   {

```

```

       my $read_load_val = $$Mod {read_min_wait_states} - 1;
       my $write_load_val = $$Mod {write_min_wait_states} - 1;

```

```

55       $read_counter_mux_string .= "$$Mod {internal_active_signal} -->
$read_load_val,"
       if ($read_load_val >= 0);

```

```

       $write_counter_mux_string .= "$$Mod {internal_active_signal} -->
$write_load_val,"
       if ($write_load_val >= 0);

```

```

60   }

```

```

   &Modified_Mux ($read_counter_mux_string);

```

09830106 "061201

&Modified_Mux (\$write_counter_mux_string);

&Mux (" \$counter_mux_string, declare=1,
 (~\$sys_writen) --> write_counter_load_value,
 (~\$sys_readn) --> read_counter_load_value,
 ");

#####

Wait-state counter.

#

There are two interesting questions about this counter:

#

-- When should it load?

-- When should it count?

#

**** When to load:

#

This counter gets loaded at the beginning of every bus-cycle

for which a fixed wait is required. We already have a signal

which tells us when it's the beginning of a bus-cycle, so we

need some additional logic to tell us if a fixed-wait is required.

We build that logic herein below, before the counter proper.

#

**** When to count:

#

This counter "sticks" at zero. This makes the whole scheme

much easier to manage/understand. Consequently, the counter

is only enabled when its current value is nonzero.

#

The only other thing that can stop this counter from a-countin'

is a wait-request from the active peripheral. This

has the effect of extending the current Bus Cycle-- setup-time,

hold-time, and all.

#

Note that there is a question of interpretation here: Do

we want to allow a peripheral to extend a bus-cycle during its

own setup-time? Or should we not listen to it until we actually

issue it an active read/write strobe? This is largely a question

of definition. I chose to -always- listen to peripheral-controlled

waits, even during setup/hold periods. Because it makes the logic

easier. So there.

#

my @fixed_wait_active_terms = ("1'b0"); # 1'b0 fixes empty-strings.

foreach \$Mod (&Get_Sys_Slave_List(\$Sys))

{

 if (\$\$Mod(read_min_wait_states) > 0) {

 push (@fixed_wait_active_terms,
 "((~\$sys_readn) && \$\$Mod(internal_active_signal))");

 }

 if (\$\$Mod(write_min_wait_states) > 0) {

 push (@fixed_wait_active_terms,
 "((~\$sys_writen) && \$\$Mod(internal_active_signal))");

 }

}

This signal stays true during the entire bus-cycle

for which a fixed (counter) wait-state applies.

#

&PBM_Wire ("fixed_wait_active", 1, join (" || ", @fixed_wait_active_terms));

```

# The counter itself, per-se.
#
# I use one call to "&Delay" to build a whole counter. Nobody else
# seems to appreciate the mighty "&Delay" function--the fools.
5 #
&PBM_Wire ("counter_ne_zero");
&PBM_Wire ("wait_count_enable", 1, "(~periph_wait_asserted) &&
                                         (counter_ne_zero)");

10 &PBM_Wire ("bus_cycle_start"); # declared here, computed later.
&PBM_Wire ("wait_load_enable", 1, "bus_cycle_start &&
                                         fixed_wait_active");

    # Optimization: Avoid emitting the counter-register if the maximum
15 # counter load-value happens to be zero. This often happens
    # when we have a bunch of one-wait-state peripherals in the
    # system. In this case, the single wait-state is controlled
    # by the "wait_load_enable" signal, and we can do away with the
    # counter per-se in its entirety. A clever synthesis tool might
20 # figure this out on its own, but...why risk it?
if ($max_counter_value <= 1) {
    &PBM_Wire ("wait_counter", 1, "1'b0");
} else {
    &Delay ("out      = wait_counter,
25         in        = (wait_counter - 1),
         enable     = (wait_load_enable || wait_count_enable),
         sync_set   = wait_load_enable,
         set_value  = counter_load_value,
30         width     = $wait_counter_bits,
         reset      = ,
         ");
}
&PBM_Assign ("counter_ne_zero", "wait_counter != $wait_counter_bits\'b0");
35 &PBM_Wire ("fixed_wait_asserted", 1, "counter_ne_zero || wait_load_enable");

#####
# Registered chip-selects.
#
40 # This is a nuisance, but at least it's easy.
#
# While we're listing all the reasons to extend a bus cycle,
# let us not forget registered chip-selects. They blow!
#
45 # Whenever the registered (delayed) version of a chip-select
# differs from the current (un-delayed) version, we view
# this as a bad thing. We extend the current bus cycle until
# the signals agree (i.e. we wait 1 clock).
#
50 my @reg_select_wait_terms = ("1'b0"); # Ah, the old "1'b0" trick.

foreach $Mod (&Get_Sys_Slave_List($Sys))
{
    next unless $$Mod{Uses_Registered_Select_Signal};
55
    # Modules -might- have more than one registered select signal,
    # but we don't really care. They all have the same time-behavior.
    #
    my $reg_signal = &Get_Sys_Signal ($Mod, "registeredselectn");
    push (@reg_select_wait_terms, "((~$reg_signal)
60 $$Mod{internal_active_signal}))");
}

```

```
&PBM_Wire ("reg_select_wait_asserted", 1,
           join (" || ", @reg_select_wait_terms));
```

```
#####
```

```
# Bus-Cycle Status.
```

```
#
```

```
# One of the major responsibility of the wait-state generator
# is to tell the rest of the world (not the least, the CPU)
# where we are in the bus cycle: Beginning, middle, or end.
```

```
#
```

```
# So. How do we know when a bus cycle begins? The only thing
# we know for sure is that one bus cycle begins when another one
# ends--it's like a Zen koan, isn't it?
```

```
#
```

```
# But seriously. A bus-cycle always ends when there is no longer
# any reason to extend it. Said more prosaically, the signal
# "extend_bus_cycle" will always be zero during the last clock
# of any bus-cycle.
```

```
#
```

```
# So, our only means for deciding that this is the -beginning- of
# a cycle is to notice that "extend_bus_cycle" was false (0).
```

```
#
```

```
# -- Bonus reason.
```

```
# Well, there's another way, too, that we can tell if this is
# the start of a bus-cycle: If the CPU (master) is -idle-
# (both read-strobe -and- write strobe inactive). Masters
# sometimes do this. If so, we consider that whole long
# time the "start" of the bus-cycle.
```

```
#
```

```
# **** Reasons To Extend.
```

```
#
```

```
# There are three "ordinary" reasons to extend a bus-cycle, and
# one "special" one. Let's start with the ordinary (slave_driven) reasons:
```

```
#
```

- # 1) The peripheral, itself, can request wait-states.
- # 2) We can have a fixed (counted) number of wait-states,
due to both traditional wait-states and setup/hold times.
- # 3) That whole sorry business about registered chip-selects.

```
#
```

```
# And here's the "special" reason
```

```
#
```

```
# *4) Data has to percolate through the principal databus drive-back
# register.
```

```
#
```

```
# But the only way we know when it's time to do (*4) is when
# the bus-cycle is no longer extended for any of the other reasons (1..3).
# But (*4) still counts as extension of the current bus-cycle.
# Therefore, we have to account for (1..3) and (4) separately.
# That's OK--I'm an amateur accountant.
```

```
#
```

```
#
```

```
# As long as the master is just sittin' there, we take that to
# mean a cycle "is in the process of starting".
```

```
#
```

```
&PBM_Wire ("master_is_idle", 1,
           "(. &Get_Sys_Signal($$Sys{master}, "readn" ).)" && "
           "(. &Get_Sys_Signal($$Sys{master}, "writen" ).)" );
```

```
#
```

```
&PBM_Wire ("slave_wait_asserted", 1, "
           reg_select_wait_asserted ||
           fixed_wait_asserted      ||
           periph_wait_asserted     ");
```

```
#
```

```
# The slave gets its chance to wait, or talk, or whatever. Then
```

09880106.061201

```

# it's over--for good.
# Orion hates &Delay because he never knows which of sync_set or
# sync_reset or any of the other options gets priority.
# reg slave_phase is defined above because the outgoing chip
# selects need to be gated by it.

5  &Vprint("
always @(posedge clk)
begin
10     if ((~reset_n) || (bus_cycle_end))
        begin
            slave_phase <= {1 {1'b1}};
        end
    else begin
15         if (~slave_wait_asserted & ~master_is_idle) begin
            slave_phase <= {1 {1'b0}};
        end
    end
end
20 ");

# Don't extend a cycle when the master is idle!
&PBM_Wire ("extend_bus_cycle", 1,
    "(slave_wait_asserted && slave_phase) ||
    (data_driveback_asserted
    ) ");

# And a bunch of different names for the same thing (used elsewhere):
#
&PBM_Assign ("bus_cycle_end",      "~extend_bus_cycle && ~master_is_idle");
30 &PBM_Assign ("bus_cycle_data_ready", "bus_cycle_end" );

# Generate a privately-named version, then assign to global wire:
&Delay ("out      = bus_cycle_start_reg,
35     in        = bus_cycle_end,
        sync_set = master_is_idle,
        set      = reset,
        ");
&PBM_Assign ("bus_cycle_start", "bus_cycle_start_reg");
40

#####
# Data drive-back timing
#
45 # We are responsible for coming up with this nasty-little signal
#
#     -- data_driveback_asserted.
#
# this is true when:
50 #
#     1) This is a read-cycle.
#
#     2) The currently-selected peripheral is -not- on the
#         "principal" databus
55 #
#     3) There is no longer any other reason to extend the current
#         bus cycle (slave_wait_asserted is zero).
#
#     4) We're not still gathering-up a data-value as part of a
60 #         multiple-bus-cycle dynamic-bus-sizing operation.
#
#         ... and, lest we forget (which we did)...
#
#

```

09030105 "051201

```

#      5) "data_driveback_asserted" wasn't true last time, otherwise
#      conditions (1)..(4) above will persist forever, and the
#      master will hang-up indefinitely.
#
5 # First, come up with a signal that tells us about (2):
#
my @driveback_active_terms = ("1'b0");
if ($$Sys{Principal_Tri_State_Data_Bus}) {
  foreach $Mod (&Get_Sys_Slave_List ($Sys)) {
10     if (($$Mod{Uses_Tri_State_Data_Bus}) &&
        ($$Mod{Tri_State_Data_Bus} eq $$Sys{Principal_Tri_State_Data_Bus}))
        {
            next;
        }
15     push (@driveback_active_terms, $$Mod{internal_active_signal});
    }
}

20 &Delay ("out = data_driveback_last_time,
        in  = data_driveback_asserted,
        reset = ,
        ");

&PBM_Assign ("data_driveback_active", join(" || ", @driveback_active_terms));

25 &PBM_Assign ("data_driveback_asserted", "
        data_driveback_active      &&
        (~$sys_readn)              &&
        (~slave_wait_asserted)     &&
30     (~dbs_wait_asserted)         &&
        (~data_driveback_last_time) ");

#####
# Transaction status
35 #
# The wait-state generator is responsible for the transaction-level
# timing as well as the bus-cycle-level timing. Recall that a transaction
# can take multiple bus cycles.
#
40 # And, once again, we are faced with the same question: How do
# we know when a transaction starts? It starts when another
# transaction ended (search for "Zen koan," above).
#
# At this time, there's only one way a transaction will keep going
45 # even when a bus-cycle has ended--if we're in the middle of a
# dynamic-bus-sizing operation.
#
&PBM_Wire ("extend_bus_transaction", 1,
50     "extend_bus_cycle || dbs_wait_asserted");

# And a bunch of different names for the same thing (used elsewhere):
#
&PBM_Wire ("bus_transaction_end", 1, "~extend_bus_transaction");

55 # Generate a privately-named version, then assign to global wire:
&Delay ("out    = bus_transaction_start_reg,
        in      = bus_transaction_end,
        set     = reset,
        ");
60 &PBM_Assign ("bus_transaction_start", "bus_transaction_start_reg");

# Lookie here: The --FINAL-- wait-request signal to the master:
#

```

09380106 "06:12:01

```

    &PBM_Assign (&Get_Sys_Signal ($Sys{master}, waitrequest),
        "extend_bus_transaction");
}

5  #####
# PBM_Emit_Setup_Hold_Logic
#
# Given a %Sys-hash (reference), emits all the logic to implement
# "custom" readn / writen-strobes.
10 #
# **** Custom Read / Write strobes
#
# If your module requests setup- or hold-times, then it
# gets its own private readn- or writen-signal, which is only
15 # active for a sub-portion of the bus cycle (unlike the masters'
# control signals, which are asserted for the entire cycle).
#
# Each "custom strobe" is generated by an S-R flip-flop. We
# need to assert the strobe at a particular point in the cycle,
20 # then deassert it at some later point. We use the counter-value
# (and the load-counter signal) to determine when to assert/deassert
# the custom write-strobe (i.e. when to set/reset the flip-flop).
#
# We can only use this scheme because we previously guaranteed that
25 # all "custom" strobes have at least one clock's worth of setup time.
# This gives us enough time to compute the S/R inputs to the
# custom-strobe register. (see "Funny setup/hold rule" in
# &Check_Avalon_Rules).
#
30 #####
sub PBM_Emit_Setup_Hold_Logic
{
    my ($Sys) = (@_);
    &Emit_Comment ("\n Setup- / Hold-Time Logic \n");

35
    #####
    # First, do full-clock setup/hold times.
    #
    # previous rule-checks guarantee no overlap with fractional case.
    # But check anyway.
    #
    foreach $Mod (&Get_Sys_Slave_List ($Sys))
    {
45        # All the modules we care about will have setup-times.
        next unless $$Mod{Setup_Time} > 0;
        &Debug (0, "Generating setup/hold-logic for $$Mod{name}");

        $$Mod{Hold_Time} =~ /half/ or die "
50         Module $$Mod{name}: internal hold-time consistency check violation";

        # Come up with an expression that says when to
        # assert the strobe (separate for read- and write-strobes)
        #
55        # The strobe is actually asserted on the clock -after- the
        # $read/write_assert_expr goes true, because we have to
        # propagate through the custom-strobe register.
        #
        my $master_read_expr = "(~.&Get_Sys_Signal($Sys{master}, "readn").)";
        my $master_write_expr = "(~.&Get_Sys_Signal($Sys{master}, "writen").)";
60        my @read_assert_terms = ($$Mod{internal_active_signal},
            $master_read_expr);

```

09380106-061201
102190-90T0886

```

my @write_assert_terms = ($$Mod{internal_active_signal},
$master_write_expr);

if ($$Mod{Setup_Time} == 1)
{
    # Because of the prop-delay (above) we have to order
    # our strobe on the first clock of the bus-cycle. Fortunately,
    # there's a signal just for that purpose:
    #
    10 push (@read_assert_terms, bus_cycle_start);
    push (@write_assert_terms, bus_cycle_start);
} else {
    # If we got to here, notice that Setup_Time is 2 or greater.
    # In this case, we have to use the output of the wait counter.
    15 # Recall that the wait-counter counts down, so this makes
    # the math a bit of a head-scratcher.
    #
    # Justifying this numerical expression is a bit tricky--
    # I used little text-tables to "simulate" some example
    20 # count sequences:
    #
    # Example: Tsu = 2, Th = 0, min_wait-states = 2
    #      wait-counter | expr true? | output write-strobe
    #      -----+-----+-----
    #      --load--      -          idle
    #      1             assert      idle
    #      0             deassert     ACTIVE
    #
    # Example: Tsu = 2, Th = 0, min_wait-states = 5
    #      wait-counter | expr true? | output write-strobe
    #      -----+-----+-----
    #      --load--      -          idle
    #      4             assert      idle
    #      3             -          ACTIVE
    #      2             -          ACTIVE
    #      1             -          ACTIVE
    #      0             deassert     ACTIVE
    #
    # Example: Tsu = 2, Th = 3, min_wait-states = 5
    #      wait-counter | expr true? | output write-strobe
    #      -----+-----+-----
    #      --load--      -          idle
    #      4             assert      idle
    #      3             deassert     ACTIVE
    #      2             -          idle
    #      1             -          idle
    #      0             -          idle
    #
    50 my $read_assert_count =
        $$Mod {read_min_wait_states} - 1 - ($$Mod{Setup_Time} - 2);
    my $write_assert_count =
        $$Mod{write_min_wait_states} - 1 - ($$Mod{Setup_Time} - 2);

    55 push (@write_assert_terms, "(wait_counter == $write_assert_count)");
    push (@read_assert_terms, "(wait_counter == $read_assert_count)");
}

&Debug (0, "@read_assert_terms is:", @read_assert_terms);
&Debug (0, "@write_assert_terms is:", @write_assert_terms);

60 my $read_assert_expr = join (" && ", @read_assert_terms );
my $write_assert_expr = join (" && ", @write_assert_terms);

```



```

# Judging from the tables (above), the expression for when to
# deassert the strobe is pretty easy. We still have to
# do a special-case when hold-time is zero, because testing for
# (wait_counter == 0) is problematic--it's zero at the beginning
# of the cycle!
#
# Also remember: As a matter of POLICY, we don't use any hold-
# time for read-cycles. We may want to revisit this policy later.
#
my $read_deassert_expr = "bus_cycle_end";
my $write_deassert_expr = "bus_cycle_end";
    $write_deassert_expr =
        "(wait_counter == $$Mod{hold_time_full_clocks})"
        if ($$Mod{hold_time_full_clocks} > 0);

# One of these days, we're actually going to have to build
# the strobe-output registers. Recall that readn/written strobes
# are logic-negative (assert --> 0). Provide an asynchronous
# "set" so they power-up inactive.
#
# FUTURE: Make these "firm flip-flops" if we determine that
# these are off-chip.
#
my $readn_strobe = &Get_Sys_Signal ($Mod, "readn");
&Delay ("out      = $readn_strobe,
        sync_reset = ($read_assert_expr),
        sync_set   = ($read_deassert_expr),
        set        = reset,
        ") if $readn_strobe;

my $written_strobe = &Get_Sys_Signal ($Mod, "written");
&Delay ("out      = $written_strobe,
        sync_reset = ($write_assert_expr),
        sync_set   = ($write_deassert_expr),
        set        = reset,
        ") if $written_strobe;

} # End: foreach $Mod...

#####
# Now generate narrow write-strobes where a fractional hold-time
# was requested.
#
my $has_fractional_hold_times = 0;
# logic-positive "name" for master strobe, to ease understanding:
#
my $master_write = "(~".&Get_Sys_Signal($$Sys{master}, "written").")";

my $made_do_shorten_write_strobe = 0;
foreach $Mod (&Get_Sys_Slave_List ($Sys))
{
    next unless $$Mod{Hold_Time} =~ /half/;
    &Debug (0, "Generating narrow write-strobe for module $$Mod{name}");
    $has_fractional_hold_times++;

    $$Mod{Setup_Time} == 0 or die "
        Module $$Mod{name}: internal hold-time consistency check violation";

    if (!$made_do_shorten_write_strobe)
    {
        &PBM_Wire ("do_shorten_write_strobe"); # forward-declared
        #here.
        $made_do_shorten_write_strobe = 1;
    }
}

```

09880106-061201

09830106-061201
TOTAL90 "GOT08265"

```

    }
    &PBM_Assign (&Get_Sys_Signal ($Mod, "writen"),
        "~($master_write && ~do_shorten_write_strobe)");
}

5
if (($has_fractional_hold_times)) {
    &Emit_Comment (" A little dab of shortening for the write-pulse");
    &Delay ("out      = write_second_half,
10         in        = $master_write,
         edge       = negedge,
         reset      = reset,
        ");
    &PBM_Assign ("do_shorten_write_strobe",
        "write_second_half && bus_cycle_end");
15 }
}

#####
# Generate_PBM
20 #
# Given a %Sys-hash (reference), we do everything required to
# generate a PBM, including opening the output-file, blorting
# all logic into it, and subsequently closing the output file
# again.
#
25 #####
sub Generate_PBM
{
    my ($Sys) = (@_);

    # Open the output file:
    #
    &PBM_Progress ("Creating PBM module: $$Sys{pbm_file}.");
    open (ALTERA_FILEHANDLE2, "> $$Sys{pbm_file}") or die "can't open
35 $$Sys{pbm_file}: $!";
    my $old_out = select(ALTERA_FILEHANDLE2);
    print "$GLOBAL_COPYRIGHT_NOTICE\n";

    &Emit_Module_Header ($$Sys{pbm_name});
40
    &PBM_Emit_Simple_Assignments ($Sys);
    &PBM_Emit_IRQ_Prioritizer ($Sys);
    &PBM_Emit_Address_Decoder ($Sys);
    &PBM_Emit_Read_Data_Mux ($Sys);
45 &PBM_Emit_Dynamic_Bus_Sizer ($Sys);
    &PBM_Emit_Wait_State_Generator ($Sys);
    &PBM_Emit_Setup_Hold_Logic ($Sys);

    &Vprint ("\n endmodule // $Sys{pbm_name}\n");
50
    close (ALTERA_FILEHANDLE2);
    select ($old_out);

    my $synth_list_ref = $$Sys{synth_file_list};
55 push (@$synth_list_ref, $$Sys{pbm_file});
}

#####
# Generate_Core
60 #
# Given a %Sys-hash (reference), we do everything required to
# generate a system-level "core" module, including opening the output-file,
# blorting all logic into it, and subsequently closing the output file
```

```

# again.
#
#####
5 sub Generate_Core
{
    my ($Sys) = (@_);

    # Open the output file:
    #
10    &PBM_Progress ("Creating Core module: $$Sys{core_file}.");
    open (ALTERA_FILEHANDLE , "> $$Sys{core_file}")
        or die "can't open $$Sys{core_file}: $!";
    my $old_out = select(ALTERA_FILEHANDLE );
    print "$GLOBAL_COPYRIGHT_NOTICE\n";

15    &Emit_Module_Header      ($$Sys{core_name});
    &Core_Emit_Wire_Declarations ($Sys);
    &Core_Emit_Instances      ($Sys);
    &Vprint ("\n endmodule // $$Sys{core_name}\n");

20    close (ALTERA_FILEHANDLE);
    select ($old_out);

    my $synth_list_ref = $$Sys{synth_file_list};
    push (@$synth_list_ref, $$Sys{core_file});
25 }

#####
# Generate_Wrapper
#
30 # Given a %Sys-hash (reference), we do everything required to
# generate a system wrapper-module, including opening the output-file,
# writing its contents, and subsequently closing the output file
# again.
35 #
# This function actually does a very simple thing: Create a
# wrapper-module with a single black-box instance in it. It looks
# scary and complicated because it's tri-lingual, and the language
# constructs to do this simple thing are more structurally different
40 # than reason would dictate.
#
#####
sub Generate_Wrapper
{
45    my ($Sys) = (@_);

    # Open the output file:
    #
    &PBM_Progress ("Creating Wrapper module: $$Sys{wrapper_file}.");
50    open (WRAPOUT, "> $$Sys{wrapper_file}") or die
        "can't open $$Sys{wrapper_file}: $!";
    my $old_out = select(WRAPOUT);

    my $core_instance_name = "the_$$Sys{core_name}";

55    &Emit_Top_Comment ($$Sys{name}, $$Sys{hdl_language});

    # AHDL is funny, and requires that we do a semi-C-like "extern"
    # (FUNCTION) declaration of the thing we're about to instantiate:
60    #
    &Declare_Ports_For($$Sys{core_name}, 0, $$Sys{hdl_language})
        if $$Sys{hdl_language} =~ /^ahdl/i;

```

```

&Emit_Module_Header ($$Sys{name}, $$Sys{hdl_language}, 1);

# Some languages require a little opening dance before the
# instantiation:
5 if ($$Sys{hdl_language} =~ /^vhdl/i) {
    &Emit_VHDL_Component ($$Sys{core_name});
    &Vprint ("BEGIN\n");

10 } elsif ($$Sys{hdl_language} =~ /^ahdl/i) {
    &Vprint ("VARIABLE\n");
    &Vprint (" $core_instance_name : $$Sys{core_name};\n\n");
    &Vprint ("BEGIN\n");
}

15 # The instantiation per-se:
&Instantiate_And_Connect ($$Sys{core_name},
                          $core_instance_name, "", 0,
20                          $$Sys{hdl_language});

# And then, of course, everyone requires a little closing-dance:
if ($$Sys{hdl_language} =~ /^vhdl/i) {
    &Vprint ("END behavior;\n");

25 } elsif ($$Sys{hdl_language} =~ /^ahdl/i) {
    &Vprint ("END;\n");

} elsif ($$Sys{hdl_language} =~ /^verilog/i) {
30 &Emit_Comment ("\n exemplar attribute $core_instance_name NOOPT TRUE \n");
    &Vprint ("\n // synopsys translate_on \n");
    &Vprint ("\n endmodule // $$Sys{name}\n");
}

35 close (WRAPOUT);
select ($old_out);
}

#####
40 # Generate_Inc_File
#
# If we're generating an AHDL wrapper, then it's customary to
# also make an ".inc"-file. This is, I suppose, the AHDL-equivalent
# of a C-language header (".h") file.

45 #
# This function generates such a wrapper for the system if
#
#####
sub Generate_Inc_File
50 {
    my ($Sys) = (@_);

    if ($$Sys{hdl_language} !~ /^ahdl/i) {
        warn ("Suspicious call to Generate_Inc_File.
55         (language is $$Sys{hdl_language}, not AHDL)");
        return;
    }

    $$Sys{inc_file} = "$$Sys{system_directory}/$$Sys{name}.inc";

60 &PBM_Progress ("Creating AHDL include-file: $$Sys{inc_file}.");
    open (INCOUT, "> $$Sys{inc_file}") or die
        "can't open $$Sys{inc_file}: $!";

```

09880106-061201

```

my $old_out = select(INCOUT);

&Emit_Top_Comment ($$Sys{name},    $$Sys{hdl_language});
&Declare_Ports_For($$Sys{name}, 0, $$Sys{hdl_language});

```

```

close (INCOUT);
select ($old_out);
}

```

```

#####
# Generate_Symbol
#

```

```

# Given a ref to the %Sys-hash, this routine generates a Quartus
# BSF-file (schematic symbol) suitable for representing %Sys in a
# schematic. The actual work of symbol-layout and geometry is done
# by Aaron Ferrucci's mighty "&Generate_BSF" function in the library
# module "mk_bsf.pm".
#

```

```

# We get to say how the ports are arranged by building a list of
# port-descriptions. We hand this off to Aaron, and he gives us back
# a string, which is the entire contents of a valid BSF-file (not yet
# created). We then create the file, write Aaron's string into it,
# and we're done.
#

```

```

#####

```

```

####
# Segment-ordering
#

```

```

# The top-to-bottom order that the symbol-segments is a little
# bit important, and there are many subtle issues, both subjective
# and practical, that come into play. One could write an entire
# program, I'm sure, just fiddling-around with the perfect ordering.
# But it's pretty clear that whatever hash-ordering comes out of
# "keys" is probably not what the user wanted.
# I'm going to sort thus:
#

```

```

# -- If you're the master, you're first.
# -- Any module with "use_tri_state_bus" takes priority.
# -- alphabetical by class-name
# -- alphabetical by module name.
#
# We're sorting a list of %Mod-refs, so $a and $b are like "$Mod"
#

```

```

sub segment_sort
(

```

```

    return -1 if $$a{Is_Bus_Master};
    return 1 if $$b{Is_Bus_Master};

```

```

# No one was the master. The guy with the tri-state databus wins.
return -1 if $$a{Uses_Tri_State_Data_Bus} && !$$b{Uses_Tri_State_Data_Bus};
return 1 if !$$a{Uses_Tri_State_Data_Bus} && $$b{Uses_Tri_State_Data_Bus};

```

```

# It's a tie! Whoever has the alphabetically-first class-name wins:
my $class_result = $$a{class} cmp $$b{class};
return $class_result if $class_result != 0;

```

```

# It's still a tie! Whoever has the alphabetically-first module name wins:
return $$a{name} cmp $$b{name};

```

```

)

sub Generate_Symbol
(

```

```

my ($Sys) = (@_);

&PBM_Progress ("Generating Symbol $$Sys{name}.bsf");

5  # There will -always- be a group at the top with the unwavering
   # clk/reset ports on it.
   #
   my @symbol_segments = ("clk      | 1 | input,
                           reset_n | 1 | input,");

10  # It sure would be nice if there were, say, a database of all the
   # widths and directions of every port on the system-module.
   #
   # HEY! It's our lucky day!
   # Someone already did "&List_Ports_For" on the system-module. Get
15  # handy hashes of directions and widths:
   #
   my $width_hash = &Get_Port_Widths      ($$Sys{name});
   my $dir_hash   = &Get_Port_Directions ($$Sys{name});

20  # Put all shared-port groups first:
   foreach $bus_name ($$Sys{tri_state_bus_list}) {
       my @bus_ports = ();
       my $shared_port_table = $$Sys{shared_port_table};
       foreach $shared_port_name (keys(%$shared_port_table)) {
25         my $w = $width_hash{$shared_port_name};
         my $d = $dir_hash  {$shared_port_name};
         push (@bus_ports, "$shared_port_name | $w | $d");
       }
       # A null string here should not happen, but don't risk it:
       my $segment_description = join ("", "\n", @bus_ports);
       push (@symbol_segments, $segment_description) if $segment_description;
   }

35  # Now go through the system, module-by-module, and make
   # a segment for each one's external (non-shared) ports.
   # Note that we even include the master. Nioses don't have any
   # external ports, but who knows what the future may bring?
   #
40  foreach $Mod (sort segment_sort &Get_Sys_Module_List($Sys)) {
       my @mod_ports = ();

       foreach $Port (&Get_Module_Port_List($Mod)) {
           next if      $$Port{is_shared};
           next unless $$Port{is_external};

           my $pin_name = $$Port{system_signal};
           my $w         = $width_hash{$pin_name};
           my $d         = $dir_hash  {$pin_name};
50         push (@mod_ports, "$pin_name | $w | $d");
       }
       # Avoid asking for a "segment" for modules that don't have any ports:
       my $segment_description = join ("", "\n", @mod_ports);
       push (@symbol_segments, $segment_description) if $segment_description;
55  }

   # Mr. Ferrucci, if you please...
   #
   my $bsf_contents = &Generate_BSF ($$Sys{name}, @symbol_segments);

60  # Do the usual file-opening mechanics:
   #
   my $bsf_filename = "$$Sys{system_directory}/$$Sys{name}.bsf";

```

09330105 "061201
TOTAL

```

open (BSFOUT, "> $bsf_filename") or die
    "Generate_Symbol: Couldn't open $bsf_filename for output. $!";
print BSFOUT $bsf_contents;
close BSFOUT;

```

5)

```

#####
# Sys_Gen_Housekeeping
#

```

10 # Well, this is a fine how-do-you-do, isn't it?

```

#
# There are a handful of annoying little things we need to do
# to prepare for the great, glorious business of system-generation.
#

```

15 # Perhaps most importantly, we use a lot of the handy-dandy
logic-generation routines from "generator_functions.vpp" (e.g. &Mux).
That's swell, but this here file (that you're reading) is a
Perl -module- which gets "use"d. That's pretty civilized, and
there are several good reasons for doing it that way:

20 #
* Perl line-numbers and error-reporting actually work.

* We don't have to "eval" this file TWICE through Vpp's
two-pass process.

25 # But, since this logic-generation program doesn't get Vpp'd,
how can we be sure that "generator_functions.vpp" ever got loaded?
Well, we can be sure by -loading it ourselves-. We do so
by calling &Vpp on our own behalf--not to generate logic, but
30 # just to load a library of handy Perl functions. Truth
is stranger than fiction.

35 # And that's not all. All of the generator-functions have an
oddball output-behavior: They don't actually print anything into
the output file unless a special vpp-variable says it's "safe".
That variable is "\$vpp_pass", which must be '2' for any of the
generator-functions to print. So, sneakily, we set this
global variable here, after we've read-in the library using
&Vpp. Because this is sneaky, it's also implementation-dependent
40 # and error-prone. This will break some day when somebody
changes the way Vpp works inside, I guarantee it. Global variables
truly are evil.

45 # FUTURE WORK:

I respectfully suggest that the approach used by "generator_functions.vpp"
could use a good overhaul. Phrasing all that stuff as a real Perl
module would have several advantages--not the least of which would
be eliminating the need for this kludge you see here.

50 #

sub Sys_Gen_Housekeeping

{
 my (\$Sys) = (@_);
55
 my @Synth_File_List = ();
 \$\$Sys{synth_file_list} = \@Synth_File_List;

 \$PBM_VERBOSE = \$\$Sys{verbose}; # More evil globals.

60
 # run Vpp, but just to "import" the generator-functions library:
 &Vpp ("-Q",
 "-D", \$\$Sys{system_directory},

09830105.061201

```

        "-H", "$$Sys{sopc_directory}/bin/vpp/generator_functions.vpp"
    );

    # Aren't I sneaky?
    #    !!BANG!! ... Ouch! My foot!
    #
    $vpp_pass = 2;    # this lets all the vpp-libraries actually print.
}

10 #####
   # Generate_PBM_And_System
   #
   # Given a reference to the system-level PTF section (and
   # an %arg-hash that we inherit from mk_systembus), this
15   # function does everything necessary to:
   #
   #    * Create an HDL implementation of the system PBM
   #    * Create an HDL implementation of the system-level "core" module.
   #    * Create an HDL wrapper.
20   #
   # This task involves rummaging-through the PTF data, opening all the
   # appropriately-named files in the appropriate places, writing all the
   # HDL-code into them, and tying a bow around the whole mess.
   #
25   # Once you've done this, the only thing left for you to do is
   # synthesize it and include it in your design.
   #
   # This function returns (by reference) the constructed database
   # known as "%Sys." This way, our caller can snoop on all the
30   # excellent work we've done on his behalf.
   #
   #####
   sub Generate_PBM_And_System
   (
35       my ($arg, $db_Sys) = (@_);

       my %Sys; undef %Sys;
       %Sys = %$arg;          # Start off by knowing everything that
                               # mk_systembus knows.
40
       &Sys_Gen_Housekeeping    (\%Sys);

       &Get_System_Data_From_PTF (\%Sys, $db_Sys);
       &Check_Avalon_Rules      (\%Sys);
45       &Create_System_Port_Lists (\%Sys);

       &Generate_PBM            (\%Sys);
       &Generate_Core           (\%Sys);
       &Generate_Wrapper        (\%Sys);
50       &Generate_Inc_File       (\%Sys)    if $Sys{hdl_language} =~ /^ahdl/i;
       &Generate_Symbol         (\%Sys);

       return (\%Sys); #caller may want to know new info too;
55   }

   1;    # Perl modules have to say "1".

60

```


ptf_parse.pm

#use strict; # dont check in with enabled

5

Each Section consists of a
Type, a Name (which can be
be blank), and an unordered
list of Elements.

10

Each Element can be either
a Name and a Value, or another
Section.

15

An Element list is an associative
array where the key is the name of
a value, or the name of a module kind.

20

For a value, the value at that key is just the value.

For a module kind, the value is another associative
array whose keys are the module names, and whose
contents are each element lists.

25

PTFHash ->

{
name => name of assignment, or kind of section if section, or name of
file if outer

30

data => value of assignment, or name of section if section
kind => "section" or "file" (or nothing means "assignment")

section => PTFHash if section

}
#

35

arrayRef ptf_ReadSectionElements(string)

40

returns a reference to
an array of elements.

sub ptf_ReadSectionElements

45

{
 my \$data = shift;
 my @elementList;

 while(\$data)

50

 {

 #
 # Find the next thing, which is either
 # a name="value"; pair, or a SECTION name {} block.
 #

55

 if(\$data =~

 /^ # beginning of string
 ([^\;\\{]*) # the element, up to the next ; or {
 ; # terminating semi
 (.*) # and the whole rest of the string.
 \$/sx)

60

 {
 #

09880105-051201

```

# It's a value, since it ends with a semicolon
#

my $elementLine;
my $elementName;
my $elementData;

$data = $2;
$elementLine = $1;

if($elementLine =~ /^\\s*(.*?)\\s*=\\s*"(.*)"\\s/)
{
    my $newElement;

    $elementName = $1;
    $elementData = $2;

    $newElement{name} = $elementName;
    $newElement{data} = $elementData;
    push(@elementList,\\$newElement);
}

}

elseif($data =~
/^                # beginning of string
([^;\\{]*)        # the element, up to the next ; or {
\\{                # section beginning
(.*)              # and the whole rest of the string.
$/sx)

{
#
# It's a section, since it ends with a left-curly
#

my $sectionKind;
my $sectionName;
my $sectionKindName;
my $sectionContents;

$data = $2;
$sectionKindName = $1;

if($sectionKindName =~ /^\\s*(.*?)\\s*$/s)
{
    $sectionKind = $1;
    $sectionName = "";

    #
    # See if the section has a name or no...
    #

    if($sectionKind =~ /^\\s*(\\S?)\\s+(\\S?)\\s*$/s)
    {
        $sectionKind = $1;
        $sectionName = $2;
    }

    $sectionKindName = $sectionKind . " " . $sectionName;

    #
    # Find matching curlybrace ending
    #
    {

```

09880106-061201
T02T90"90T08860

```
5      my $braceDepth = 1;
      my $dataLen = length($data);
      my $i;
      my $ch;
      my %newElement;

      $i = 0;
      while ($i < $dataLen and $braceDepth > 0)
      {
10         $ch = substr($data,$i,1);
         $braceDepth++ if($ch eq "{");
         $braceDepth-- if($ch eq "}");
         $i++;
      }

15     # Bust in half, neither half gets the last '}'
     $sectionContents = substr($data,0,$i-1);
     $data = substr($data,$i);

20     #
     # Recursively descend, and get the tree
     # for this section, and add it to the list.
     #

25     $newElement{name} = $sectionKind;
     $newElement{data} = $sectionName;
     $newElement{section}

     ptf_ReadSectionElements($sectionContents);

30     push(@elementList,%newElement);
     }
   }
   else
35   {
     $data = "";
   }
40 }

#
# Return reference to this whole tree.
#

45 return \@elementList;
}

# -----
# ptf new_ptf_from_file
50 #
# returns a ptf hashref

sub new_ptf_from_file
{
55   my $fileName = shift;
   my $fileContents;
   my %ptfHash;

   $fileContents = readFile($fileName);
60   return "" if($fileContents eq "");

   # strip comments completely
   $fileContents =~ s/\s*#.*/\n\n/sg;
```

T02T90" 90T08850

```
#
# Populate top level of the ptf-ref
#
5      $ptfHash{name} = $fileName;
      $ptfHash{kind} = "file";
      $ptfHash{section} = ptf_ReadSectionElements($fileContents);

10     return \%ptfHash;
    }

# -----
# ptf new_ptf_file(fileName)
#
15     # returns a ptf hashref

sub new_ptf_file
    {
20         my $fileName = shift;
        my \%ptfHash;

        $ptfHash{name} = $fileName;
        $ptfHash{kind} = "file";

25         return \%ptfHash;
    }

# -----
# ptf new_ptf(name,data)
#
30     # returns a ptf hashref with the
    # new assignment ptf. (It becomes
    # a "section" if you add any children
35     # to it.)

sub new_ptf
    {
40         my \%ptfHash;
        my $name = shift;
        my $data = shift;

        $ptfHash{name} = $name;
        $ptfHash{data} = $data;
45         return \%ptfHash;
    }

# -----
# get_child_count
#
50     sub get_child_count
        {
            my $ptfRef = shift;

55             my $childCount = 0;
            my $sectionRef = $$ptfRef{section};

            $childCount = $$sectionRef + 1;
            return $childCount;
60         }

sub get_data
    {
```

```

        my $ptfRef = shift;

        return $$ptfRef{data};
    }

5   sub get_name
    {
        my $ptfRef = shift;

10      return $$ptfRef{name};
    }

    sub set_data
    {
15      my $ptfRef = shift;
        my $new_data = shift;

        $$ptfRef{data} = $new_data;
        return;
20    }

    sub set_name
    {
25      my $ptfRef = shift;
        my $new_name = shift;

        $$ptfRef{name} = $new_name;
        return;
30    }

    # -----
    # get_child
    #

35   sub get_child
    {
        my $ptfRef = shift;
        my $index = shift;
        my $key;
40      my $sectionRef;

        $sectionRef = $$ptfRef{section};

        return $$sectionRef[$index];
45    }

    # -----
    # get_child_by_name(ptfRef, childName, [returnIndexToo])
    #
50   # childName = "*": return first child
    # childName = "fish": return child named "fish"
    # childName = "fish blue": return child section type "fish", name "blue"
    #

55   sub get_child_by_name
    {
        my $ptfRef = shift;
        my $name = shift;
        my $returnIndexToo = shift;

60      my $sectionRef;
        my $childRef;

```

```

my $nameFront;
my $nameBack;
my $i;

5      ($nameFront,$nameBack) = split(/ /,$name,2);

      if($$ptfRef{section})
      {
10         $sectionRef = $$ptfRef{section};
      }
      else
      {
15         return "";
      }

      for($i = 0; $i <= $$sectionRef; $i++)
      {
20         $childRef = $$sectionRef[$i];

         # easy case
         last if ($name eq "");

         # harder case for named section
         last if ($nameFront eq $$childRef{name}
25            and ($nameBack eq "" or $nameBack eq
$$childRef{data}));

         $childRef = "";
      }
30      return ($childRef,$i) if $returnIndexToo;
      return $childRef;
    }

35  # -----
  # sub get_child_by_path(ptfRef, path [, buildIfAbsent[, returnParentToo])
  #
  # path is slash-separated of names as above
  #
40  sub get_child_by_path
    {
      my $ptfRef = shift;
      my $path = shift;
      my $buildIfAbsent = shift;
45      my $returnParentToo = shift; # if true, return 2 element list

      my $pathFront;
      my $pathBack;
      my $childRef;
50      my $parentRef;
      my $index;

      # In case path is null, or nearly so
      $childRef = $ptfRef;

55      while($path ne "" and $path ne "/") # paths dont really have a leading
slash... but.
      {
60         ($pathFront,$pathBack) = split(/\/$/, $path, 2);
         ($childRef,$index) = get_child_by_name($ptfRef,$pathFront,1);
         if($childRef eq "" and $buildIfAbsent)
         {
           my %child;

```

00000106-061201
T02T90-90T00000

```

        my $sectionRef;

        # Build new child, then take its reference

5           {
              my ($childName,$childData) = split(/ /,$pathFront,2);

              $child{name} = $childName;
              $child{data} = $childData;
10             $childRef = \%child;
              }

              if(not $$ptfRef{section})
              {
15                 $$ptfRef{section} = [];
              }

              $sectionRef = $$ptfRef{section};
              push(@$sectionRef,$childRef);
              }

20         return "" if($childRef eq "");

        $path = $pathBack;
        $parentRef = $ptfRef;
        $ptfRef = $childRef;
25     }

    return ($parentRef,$childRef,$index) if $returnParentToo;
    return $childRef;
    }

30 # -----
# sub get_first_children_of_type(ptfRef, type)
#
# returns an array of all ptfRef's first children
# that have type ($type);
35 sub get_first_children_of_type
{
    my ($db_parent,$type) = @_;
    my $num_children = &get_child_count ($db_parent);

40     my @child_array;

    for ($child_index = 0;
        $child_index < $num_children;
45         $child_index++)
    {
        my $db_child = &get_child ($db_parent,$child_index);
        next if &get_name ($db_child) ne "$type";
        push (@child_array,$db_child);
50     }

    return (@child_array);
}

# -----
55 # sub delete_child(ptfRef, path)
#
# path is slash-separated of names as above
sub delete_child
{
60     my $ptfRef = shift;
    my $path = shift;
    my $childRef;
    my $parentRef;

```

09830106-061201

```
my $index;
my $sectionArrayRef;

($parentRef,$childRef,$index) = get_child_by_path($ptfRef,$path,0,1); #
5 extra flag to return parent, too

$sectionArrayRef = $$parentRef{section} if $parentRef ne "";

splice(@$sectionArrayRef,$index,1);
10 return "";
}

# -----
15 # sub get_data_by_path(ptfRef, path)
#
# path is slash-separated of names as above
#
sub get_data_by_path
20 {
    my $ptfRef = shift;
    my $path = shift;
    my $childRef;
    my $result = "";

    25 $childRef = get_child_by_path($ptfRef,$path);
    $result = $$childRef{data} if $childRef;

    return $result;
    30 }

# -----
# sub add_child_data(ptfRef, path, data)
#
35 sub add_child_data
    {
        my $ptfRef = shift;
        my $path = shift;
        my $data = shift;
        40 my $childRef;

        $childRef = get_child_by_path($ptfRef,$path,1); # build if absent
        $$childRef{data} = $data;

        45 return;
    }

# -----
# sub add_child(ptfRef, path, childPTFRef)
50 #
sub add_child
    {
        my $ptfRef = shift;
        my $path = shift;
        55 my $childPTFRef = shift;

        my $childRef;          # reference down in the path
        my $sectionRef;

        60 $childRef = get_child_by_path($ptfRef,$path,1); # build if absent

        $$childRef{section} = [] if(not $$childRef{section});
        $sectionRef = $$childRef{section};
```



```
push(@$sectionRef,$childPTFRef);
```

```
return;  
}
```

```
5  
# -----  
# toString(ptfRef[,indentLevel])  
#  
# (helper routine: sectionToString(ptfRef,indentLevel)  
10 #
```

```
sub toString
```

```
{  
    my $ptfRef = shift;  
    my $indentLevel = shift;  
    my $result = "";
```

```
    if($$ptfRef{kind} eq "file"  
        and $indentLevel eq "")
```

```
{  
    $result .= "#\n";  
    $result .= "# file: " . $$ptfRef{name} . "\n";  
    $result .= "# date: " . dateTime() . "\n";  
    $result .= "# generated by a perl script\n";  
    $result .= "#\n";  
    $result .= childrenToString($ptfRef,$indentLevel);  
}
```

```
elseif($$ptfRef{section})
```

```
{  
    $indentLevel += 3;  
    $result      .= indentPrint($indentLevel,$$ptfRef{name},"  
    ",$$ptfRef{data});  
    $result      .= indentPrint($indentLevel,"(");  
    $result      .= childrenToString($ptfRef,$indentLevel);  
    $result      .= indentPrint($indentLevel,")");  
}
```

```
elseif($$ptfRef{name} ne "")
```

```
{  
    $indentLevel += 3;  
    $result      .= indentPrint($indentLevel,$$ptfRef{name},"  
    \"\",$$ptfRef{data},\"\\\";\");  
}
```

```
    return $result;  
}
```

```
sub childrenToString
```

```
{  
    my $ptfRef = shift;  
    my $indentLevel = shift;  
    my $result = "";
```

```
    my $childCount;  
    my $i;  
    my $childRef;
```

```
    $childCount = get_child_count($ptfRef);  
    for($i = 0; $i < $childCount; $i++)
```

```
{  
        $childRef = get_child($ptfRef,$i);  
        $result .= toString($childRef,$indentLevel);  
    }
```

```
    return $result;
```

09380106-061201

```

    }

sub indentSprint
5    {
    my $indentLevel = shift;
    my $a;
    my $result;

    $result = " " x $indentLevel;
10    for($a = 0; $a < 15; $a++)
        {
            $result .= shift;
        }

15    $result .= "\n";
    return $result;
}

20 # -----

sub ptf_indent
25 {
    my $a = shift;
    my $result;

    $result = sprintf("[%02d]", $a);
    $result .= " " x $a;
30    return $result;
}

# -----
# write_ptf_file(ptfRef[,fileName])
35 #
# Write the ptf to a file, optionally to a new filename
# Return "ok" if success, "" if fail.
#
sub write_ptf_file
40 {
    my $ptfRef = shift;
    my $fileName = shift;
    my $ptfString;
    my $result;

45    $fileName = get_name($ptfRef) if !$fileName;

    $ptfString = toString($ptfRef);
    $result = writeFile($fileName, $ptfString);

50    return $result;
}

# -----
55 # ptf_PrintRef(a reference)
#
# Print an indented view of the reference
#
# This is only useful as a debugging aid
60 #

sub ptf_PrintRef
{

```

```

my $theRef = shift;
my $indent = shift;

my $refType = ref $theRef;

```

```

5  if(!$refType)

```

```

    {
        print $theRef, "\n";
        return;
    }

```

```

10  if($refType eq "HASH")

```

```

    {
        my $key;
        my $value;

        print "(HASH)\n";
        $indent++;

```

```

20  foreach $key (sort(keys(%$theRef)))

```

```

    {
        print ptf_indent($indent), $key, " = ";
        $value = $$theRef{$key};
        ptf_PrintRef($value, $indent);
    }

```

```

25  return;
    }

```

```

30  if($refType eq "ARRAY")

```

```

    {
        my $arraySize = scalar @$theRef;
        my $i;
        my $value;

```

```

35  print "(ARRAY)\n";
        $indent++;

```

```

40  for($i = 0; $i < $arraySize; $i++)

```

```

    {
        $value = $$theRef[$i];
        if(defined($value))
        {
            print ptf_indent($indent), "[${i}]: ";
            ptf_PrintRef($value, $indent);
        }
    }

```

```

    }

```

```

50  # -----

```

```

    # dateTime()

```

```

    #

```

```

55  # returns a relatively nice date & time string

```

```

    #

```

```

    sub dateTime

```

```

    {
        my ($sec, $min, $hour, $mday, $mon, $year, $yday, $isdet) =

```

```

60  localtime(time);

```

```

        $mon++;
        $year += 1900;
    }

```

09880106-061201

```

my $d = sprintf("%04d.%02d.%02d", $year, $mon, $mday);
my $t = sprintf("%02d:%02d:%02d", $hour, $min, $sec);

```

```

5     return "$d $t";
    }

```

```

# -----
# readFile(fileName)
#
10  # returns the complete file contents
#
sub readFile

```

```

    {
15     my $fileName = shift;
        my $bunch;
        my $result;
        my $did;

        if(open(FILE, $fileName))
20         {
            binmode FILE;                # Bite me, Windows! --dvb
            while(read(FILE, $bunch, 32000))
                {
25                 $result .= $bunch;
                }
            close FILE;
        }

        return $result;
30    }

```

```

# -----
# writeFile(fileName, contents)
#
35  # creates new file and writes entire
    # file contents. Return "ok" if so,
    # or "" if not.
#

```

```

40  sub writeFile
    {
        my $fileName = shift;
        my $contents = shift;
        my $did;

45         #
        # Delete existing file, if any.
        #
        unlink ($fileName) if(-e $fileName);

50         $did = open(FILE, ">$fileName");
        if($did)
            {
25                 binmode FILE;                # Bite me, Windows! --dvb
55                 print FILE $contents;
                close FILE;
                return "ok";
            }

60         return "";
    }

```

```

# -----

```

09880106 061201 102190" 90108260

09880106 "061201

```
# copyFile(sourceFile,destFile)
#
sub copyFile
{
5      my $sourceFile = shift;
      my $destFile = shift;
      my $fileContents;
      my $result;

10     $fileContents = readFile($sourceFile);
      return if !$fileContents;

      $result = writeFile($destFile,$fileContents);

15     return $result;
}

# -----
# copyDirContents(sourceDir,destDir)
20 #
# If there's any files in sourceDir,
# then ensure that destDir exists, and
# copy the files over, but don't bother
# with ones which are already there.
25 #

sub copyDirContents
{
30     my $sourceDir = shift;
      my $destDir = shift;

      my @sourceDir;
      my @destDir;
      my $file;

35     opendir (DIR,$sourceDir) or return;
      @sourceDir = readdir(DIR);
      closedir (DIR);

40     opendir (DIR,$destDir) or return;
      @destDir = readdir(DIR);
      closedir (DIR);

      foreach $file (@sourceDir)
45       {
          #
          # Skip files starting with . or underscore
          #

50         if( (!( $file =~ /\^[\. _]/ )) && ( $file ne "vssver.scc" ) )
            {
                # (allow overwriting) if( (scalar grep (/^${file}$/,@destDir)
                ) == 0 )
                {
55                     mkdir($destDir,511);
                      copyFile("${sourceDir}/${file}","${destDir}/${file}");
                }
            }
        }

60     return "ok";
}
```

```

# -----
# killDirectory
sub killDirectory
{
5   return;
   my $dir = shift;
   my @dirContents;
   my $file;
   my $fullFile;

10  return if (! (-e $dir));

   opendir (DIR,$dir) or return;
   @dirContents = readdir (DIR);
   closedir (DIR);

   foreach $file (@dirContents)
   {
20     next if($file eq "." or $file eq "..");
     $fullFile = "$dir/$file";

     if( -d $fullFile)
     {
25         killDirectory($fullFile);
     }

     else
     {
30         unlink($fullFile);
     }

   }

   rmdir($dir);
}

35 # -----
# All routines go above this line.

return 1; # modules just do this

40 # End Of File

```

09880105-061201

From: Eleanor Gilbert <Eleanor.Gilbert@Sun.COM>
To: <tmayfield@beyerlaw.com>
Date: 6/6/01 1:25PM
Subject: Filing Instructions

SUN MICROSYSTEMS, INC.
PATENT DEPARTMENT
INSTRUCTIONS FOR FOREIGN FILING

****PLEASE CONFIRM RECEIPT OF INSTRUCTIONS BY TYPING YOUR NAME
IN BOX BELOW AND SENDING THIS EMAIL BACK TO ME****

Receipt of these instructions confirmed by:
Name: _____
Date: _____

TO: Todd Mayfield

FROM: Ellie Gilbert

FOREIGN FILING DEADLINE: June 30, 2001

RE: SUN REF NO.: P4407

LAWFIRM REF.: SUN1P264

SERIAL NO.: 09/608,312

FILING DATE: 06/30/00

Per the request of Marilyn Glaubenslee, please file the patent
application referenced above with the PCT. In doing so please
designate:

All countries except the U.S.

At the appropriate time we would like to use the EP as the designated
search authority.

IMPORTANT

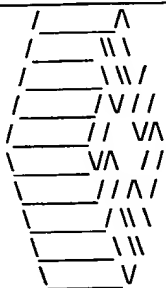
While this matter is being prosecuted, ****ALL**** foreign mail should be
directed to the Managing Sun Attorney, but address the envelope as
follows:

Patent Operations Manager
Sun Microsystems, Inc.
M/S: UPAL01-521
901 San Antonio Road
Palo Alto, CA 94303

09880106-061201

This includes the letter assigning this case to foreign associates.
Your cooperation in helping to keep our files current and accurate is appreciated.

If you have any questions concerning this matter, please call me at
(408) 343-1619.



Ellie Gilbert
Patent Specialist
Sun Microsystems, Inc.
51619 Ext.
408.343.1619 Direct
408.343.1757 Fax
eleanor.gilbert@sun.com

CC:

<foreign.docket@kat.corp.sun.com>, <eleanor.gilbert@eng.sun.com>

00000106 061201

ptf_update.pm

use ptf_parse;

```
5 #####
# PTF_Translate_Old_Version
#
# Sometimes we will find ourselves in the unfortunate position
10 # of reading a PTF-file generated by an older version of the
# kit. This function is a pre-filter which is called before anything
# else happens to the PTF-file.
#
# This script has these jobs:
15 #
# 1) Update all the signal-type names to their new version.
#
# 2) Make sure every module has the following required assignments
#    in its SYSTEM_BUILDER_INFO section:
20 #    - Data_Width
#    - Address_Width
#
# 3) Make sure every module has a correct "class" assignment at its
#    top-level.
25 #####
my %avalon_role_translation;
$avalon_role_translation {reset_n} = "resetn";
$avalon_role_translation {read_data} = "readdata";
30 $avalon_role_translation {write_data} = "writedata";
$avalon_role_translation {r_wn} = "written";
$avalon_role_translation {w_rn} = "readn";
$avalon_role_translation {be_n} = "byteenablen";
$avalon_role_translation {chip_select} = "chipselect";
35 $avalon_role_translation {wait_request} = "waitrequest";
$avalon_role_translation {bus_wait} = "waitrequest";
$avalon_role_translation {registered_chip_select_n} = "registeredselectn";
$avalon_role_translation {irq} = "irq";
40 $avalon_role_translation {address} = "address";

my %master_type_translation;
$master_type_translation {clk} = "master_input_clk";
$master_type_translation {reset_n} = "master_input_resetn";
$master_type_translation {bus_wait} = "master_input_waitrequest";
45 $master_type_translation {mem_is_32_bits} = "master_input_memis32bits";
$master_type_translation {ifetch} = "master_output_ifetch";
$master_type_translation {data_to_cpu} = "master_input_readdata";
$master_type_translation {data_from_cpu} = "master_output_writedata";
$master_type_translation {mem_addr} = "master_output_address";
50 $master_type_translation {mem_wr_n} = "master_output_writen";
$master_type_translation {mem_rd_n} = "master_output_readn";
$master_type_translation {mem_be_n} = "master_output_byteenablen";
$master_type_translation {irq} = "master_input_irq";
55 $master_type_translation {irq_number} = "master_input_irqnumber";

my %class_translation_from_1_0;
$class_translation_from_1_0 {nios} = "jnioswizard";
$class_translation_from_1_0 {uart} = "juartwizard";
$class_translation_from_1_0 {pio} = "jpiowizard";
60 $class_translation_from_1_0 {timer} = "jtimerwizard";
$class_translation_from_1_0 {onchip_rom} = "jramwizard";
$class_translation_from_1_0 {onchip_ram} = "jramwizard";
```

09880106-061201

09880106 "061201
T001

```
my %class_translation_from_jw = (
    jnioswizard      => "altera_nios",
    juartwizard      => "altera_avalon_uart",
5    jpiowizard      => "altera_avalon_pio",
    jtimerwizard     => "altera_avalon_timer",
    jramwizard       => "altera_avalon_onchip_memory",
    ji2cwizard       => "altera_avalon_i2c",
    jspiwizard       => "altera_avalon_spi",
10    jusersocketwizard => "altera_avalon_user_defined_interface",
    j2xidt71v016sawizard => "altera_nios_dev_board_sram32",
    jam29lv800b_smallwizard => "altera_nios_dev_board_flash_small",
    jam29lv800bwizard  => "altera_nios_dev_board_flash",
    jidt71v016_smallwizard => "altera_nios_dev_board_sram16",
15 );

my $PTF_CHANGES = 0;

20 sub PTF_Translate_From_Version_1_0
{
    my ($ptf_filename, $db_PTF_File) = (@_);

    my $db_Sys      = &PTF_Get_Required_Child_By_Path ($db_PTF_File, "SYSTEM");
    my $db_Sys_WSA  = &PTF_Get_Required_Child_By_Path ($db_Sys,
25                "WIZARD_SCRIPT_ARGUMENTS");

    &Rename_Old_edf_File($ptf_filename);

30    printf STDERR "****PTF_Translate_Old_Version:
        Updating old-format PTF-file: $ptf_filename\n";

    $PTF_CHANGES += &PTF_Update_Boolean_Values ($db_Sys);

35    # Get some old-tyme global parameters which we now want to poke
    # into the CPU:
    my $reset_module = &get_data_by_path($db_Sys_WSA, "reset_module");
    my $reset_offset = &get_data_by_path($db_Sys_WSA, "reset_offset");
    my $vecbase_module = &get_data_by_path($db_Sys_WSA, "vecbase_module");
    my $vecbase_offset = &get_data_by_path($db_Sys_WSA, "vecbase_offset");
40    my $mainmem_module = &get_data_by_path($db_Sys_WSA, "mainmem_module");

    &add_child_data ($db_Sys_WSA, "dtatmem_module", $mainmem_module);

45    my $num_modules = &get_child_count ($db_Sys);
    for (my $module_index = 0; $module_index < $num_modules; $module_index++)
    {
        my $db_Module = &get_child ($db_Sys, $module_index);
        next if &get_name ($db_Module) ne "MODULE";    # ignore non-modules.

50        #####
        # Blast HDL_INFO file-list assignments.
        # Any files that we need will be created by the new class's
        # generator_program.
55        #
        my $db_HDLI = &get_child_by_path($db_Module, "HDL_INFO");
        &add_child_data ($db_HDLI, "Simulation_HDL_Files", "");
        &add_child_data ($db_HDLI, "Synthesis_HDL_Files", "");

60        my $old_module_type = &get_data_by_path ($db_Module,
            "HDL_INFO/Module_Type");

        my $is_master = $old_module_type eq "nios";
```

```

my $db_Types = &get_child_by_path($db_Module, "PORT_WIRING/TYPE");
my $db_Widths = &get_child_by_path($db_Module, "PORT_WIRING/WIDTH");
my $db_WSA = &get_child_by_path($db_Module, "WIZARD_SCRIPT_ARGUMENTS");
my $db_SBI = &get_child_by_path($db_Module, "SYSTEM_BUILDER_INFO");

my $address_width = "--unknown--";
my $data_width = "--unknown--";
my $has_reg_sel = 0;
my $has_irq = 0;
my $has_tri_state = 0;

#####
# Poke-in some Nios CPU-specific parameters, if this is the CPU:
if ($is_master) {
    &add_child_data ($db_WSA, "reset_module", $reset_module);
    &add_child_data ($db_WSA, "vecbase_module", $vecbase_module);
    &add_child_data ($db_WSA, "reset_offset", $reset_offset);
    &add_child_data ($db_WSA, "vecbase_offset", $vecbase_offset);
}

#####
# Translate all the old avalon-roles into new avalon-roles.
#
my $num_ports = &get_child_count ($db_Types);
for (my $port_index = 0; $port_index < $num_ports; $port_index++)
{
    my $db_Assignment = &get_child ($db_Types, $port_index);
    my $port_name = &get_name ($db_Assignment);
    my $type = &get_data_by_path ($db_Types, $port_name);
    my $width = &get_data_by_path ($db_Widths, $port_name);

    my $new_type = $type;

    if ($old_module_type eq "nios" &&
        ($new_type =~ /(input|output)/ ))
    {
        # The old nios module had a very strange port-wiring section,
        # where no special types were given for the master's signals.
        # We have to use a special table to give the master some
        # port-types, based on the (known) port-names on the Nios:
        #
        $new_type = $master_type_translation ($port_name);
    }

    foreach $old_role (keys(%avalon_role_translation))
    {
        my $new_role = $avalon_role_translation{$old_role};
        $new_type =~ s/$old_role/$new_role/;
    }
    $data_width = $width if $new_type =~ /data$/;
    $address_width = $width if $new_type =~ /_address$/;
    $has_irq = 1 if $new_type =~ /_irq$/;
    $has_reg_sel = 1 if $new_type =~ /_registeredselectn$/;
    $has_tri_state = 1 if $new_type =~ /_data$/;

    $new_type =~ s/_shared_off_chip/_shared/;

    if ($new_type ne $type)
    {
        print STDERR "$port_name: obsolete port type $type found.\n";
        print STDERR "    automatically replaced by new type $new_type.\n";
        $PTF_CHANGES++;
    }
}

```

```

        &add_child_data ($db_Types, $port_name, $new_type) ;
    }
}

#####
# Upgrade to new PORT-section style.
#
&PTF_Update_Port_Wiring_Section($db_Module);

#####
# Use "HDL_INFO/Module_Type" assignment to figure-out new "class".
#
# We use the "Module_Type" assignment. Note that the old
# "usersocket" megawizard allowed the user to type -anything-
# into this field--so any Module_Type we don't recognize must
# be a user-socket.
#
my $class = &get_data_by_path ($db_Module, "class");
if (!$class)
{
    $class = $class_translation_from_1_0{$old_module_type};
    #JWIZ NAME CHANGE
    $class = "jusersocketwizard" if !$class;

    &add_child_data ($db_Module, "class", $class);
    print STDERR "Added 'class' assignment $class to obsolete MODULE\n";
    $PTF_CHANGES++;
}

#####
# Add absolutely-required sections to PTF:
#
&PTF_Addit_And_Warnem($db_SBI, "Data_Width", $data_width);
&PTF_Addit_And_Warnem($db_SBI, "Address_Width", $address_width);
&PTF_Addit_And_Warnem($db_SBI, "Is_Enabled", 1);
&PTF_Addit_And_Warnem($db_SBI, "Is_Bus_Master", $is_master);
&PTF_Addit_And_Warnem($db_SBI, "Date_Modified", scalar(localtime));
&PTF_Addit_And_Warnem($db_SBI, "Instantiate_In_System_Module", 1);

if ((!$is_master)) { # Checks for slaves-only:
    &PTF_Addit_And_Warnem($db_SBI,
        "Uses_Tri_State_Data_Bus", $has_tri_state);
    &PTF_Addit_And_Warnem($db_SBI,
        "Uses_Registered_Select_Signal", $has_reg_sel);
    &PTF_Addit_And_Warnem($db_SBI, "IRQ_Number", "N/A");
    &PTF_Addit_And_Warnem($db_SBI, "Has_IRQ", $has_irq);
    &PTF_Addit_And_Warnem($db_SBI, "Address_Alignment", "native");
}

#####
# Here are some module-specific updates:
#
if (($old_module_type eq "onchip_rom")) {
    # All old ROMs were 16 bits wide, so size = depth * 2
    my $size = &get_data_by_path ($db_WSA, "depth") * 2;
    my $file = &get_data_by_path ($db_WSA, "contents");
    # All old rom-files were copied to the local directory.
    # use them there:
    $file =~ s/.*?([^\s\\\/])+$/1/; # Strip-off leading path

    &PTF_Addit_And_Warnem ($db_WSA, "Writeable", 0 );
    &PTF_Addit_And_Warnem ($db_WSA, "Contents", "file");
}

```

09880106 "061201

```

&PTF_Addit_And_Warnem ($db_WSA, "Initfile", $file );
&PTF_Addit_And_Warnem ($db_SBI, "Is_Memory_Device", 1 );
&PTF_Addit_And_Warnem ($db_SBI, "Address_Span", $size );
}
5 if (($old_module_type eq "onchip_ram")) {
    # All old ROMs were 16 bits wide, so size = depth * 2
    my $size = &get_data_by_path ($db_WSA, "depth") * ($data_width/8);
    &PTF_Addit_And_Warnem ($db_WSA, "Writeable", 1);
    &PTF_Addit_And_Warnem ($db_WSA, "Contents", "blank");
10 &PTF_Addit_And_Warnem ($db_SBI, "Is_Memory_Device", 1 );
    &PTF_Addit_And_Warnem ($db_SBI, "Address_Span", $size );
}

15 if (($old_module_type eq "uart")) {
    # Uarts are the only old "printable devices".
    &PTF_Addit_And_Warnem ($db_SBI, "Is_Printable_Device", 1 );
}

20 if (($class eq "jusersocketwizard")) {
    # Whatever ports are in this PTF-file, the user gets to keep.
    &PTF_Addit_And_Warnem ($db_WSA, "keep_legacy_ports", 1);
    &PTF_Addit_And_Warnem ($db_SBI, "Tri_State_Data_Bus",
        "shared_off_chip") if $has_tri_state;
    &PTF_Addit_And_Warnem ($db_SBI, "Is_Memory_Device", 1 );
25 }

#####
# And some other oddball system-level things:
if ($freq = &get_data_by_path($db_WSA, "clock_freq")) {
30 &PTF_Addit_And_Warnem ($db_Sys_WSA, "clock_freq", $freq);
}

if ($has_tri_state) {
    &PTF_Addit_And_Warnem ($db_Sys_WSA,
        "Principal_Tri_State_Data_Bus",
        "shared_off_chip");
35 }
}

40 }

sub PTF_Translate_Old_Version
{
45 my ($ptf_filename) = (@_);

    $PTF_CHANGES = 0;

    my $db_PTF_File = &PTF_New_Required_Ptf_From_File ($ptf_filename);
50 my $db_Sys = &PTF_Get_Required_Child_By_Path ($db_PTF_File, "SYSTEM");
    my $db_Sys_WSA = &PTF_Get_Required_Child_By_Path ($db_Sys,
        "WIZARD_SCRIPT_ARGUMENTS");

#####
55 # Decide if this is an "old" PTF at-all.
# Because the version number was not in the PTF for release 1.0 of
# the kit, we have to use an heuristic: If the SYSTEM module
# doesn't have a clock frequency, then it must be old.
#
60 &PTF_Translate_From_Version_1_0 ($ptf_filename, $db_PTF_File)
    if !&get_data_by_path ($db_Sys_WSA, "clock_freq");

```

```

my $num_modules = &get_child_count ($db_Sys);
for (my $module_index = 0; $module_index < $num_modules; $module_index++)
{
    my $db_Module = &get_child ($db_Sys, $module_index);
    next if &get_name ($db_Module) ne "MODULE";    # ignore non-modules.

```

```

    my $class_name = &get_data_by_path($db_Module, "class");
    my $updated_name = $class_translation_from_jw($class_name);
    next unless $updated_name;

```

```

    &add_child_data ($db_Module, "class", $updated_name);
    $PTF_CHANGES++;
}

```

```

if ($PTF_CHANGES != 0)
{

```

```

    print STDERR "PTF_Translate_Old_Version:
    Made $PTF_CHANGES changes to obsolete PTF file $ptf_filename\n";
    &write_ptf_file ($db_PTF_File) or die "Couldn't write PTF File ";

```

```

} else {
    # print STDERR "PTF_Translate_Old_Version:  No changes made.\n";
}
}

```

```

#####
# PTF_Addit_And_Warnem
#

```

```

# Add a missing assignment to a (presumably-old) PTF-file
# if it isn't already there.  If we have to add it, then emit a
# scolding.

```

```

# Return "1" if we had to alter the PTF, "0" otherwise.
#
#####

```

```

sub PTF_Addit_And_Warnem
{

```

```

    my ($ptf_ref, $path, $value) = (@_);

```

```

    if (&get_data_by_path ($ptf_ref, $path) eq "") {
        &add_child_data($ptf_ref, $path, $value);
        my $name = &get_data ($ptf_ref);
        print STDERR "Added assignment ($path=$value) to old PTF file $name.\n";
        $PTF_CHANGES++;
        return 1;
    }

```

```

    return 0;
}

```

```

#####
# PTF_Update_Boolean_Values
#

```

```

# Recursively loops through the PTF and all its children.
# If the value of any leaf-level assignment is "TRUE" or "FALSE"
# or "yEs" or "no" or some such, then we update to the new value:
# 1/0.
#####

```

```

sub PTF_Update_Boolean_Values
{

```

```

    my ($ptfRef) = (@_);

```

```

    my $num_corrections = 0;

```

09880106 "061201

```

my $num_children = &get_child_count ($ptfRef);
for (my $child_index = 0; $child_index < $num_children; $child_index++) {
    my $child_ptf = &get_child ($ptfRef, $child_index);

5      if (&get_child_count($child_ptf) != 0) {
        $num_corrections += &PTF_Update_Boolean_Values ($child_ptf);
        next;
      }

10     # We only want to correct boolean values in SBI and WSA sections
        # (better safe than sorry)

        my $section_type = &get_name ($ptfRef);
        next unless ($section_type eq "WIZARD_SCRIPT_ARGUMENTS") ||
15         ($section_type eq "SYSTEM_BUILDER_INFO" ) ;

        my $child_name = &get_name ($child_ptf);
        my $value       = &get_data ($child_ptf);
        if (($value =~ /^TRUE$/i) ||
20         ($value =~ /^yes$/i ) )
        {
            print STDERR "Replacing $child_name value '$value' with '1'\n";
            &add_child_data($ptfRef, $child_name, "1");
            $num_corrections++;
            next;
        }
        if (($value =~ /^FALSE$/i) ||
30         ($value =~ /^no$/i ) )
        {
            print STDERR "Replacing $child_name value '$value' with '0'\n";
            &add_child_data($ptfRef, $child_name, "0");
            $num_corrections++;
            next;
        }
35     }
    return $num_corrections;
}

#####
40 # PTF_Update_Port_Wiring_Section
#
# PORT_WIRING section: how it used to be.
#
# In the old days, all the WIDTHS of the ports were together in
45 # a WIDTH section. The assignment-names were the port-names, and
# the values were the widths.
#
# All the TYPES were together in the TYPES section. The
# assignment-names were (again, and redundantly) the port-names, and
50 # the values were a twisted mutation of a little descriptor which
# contained a number of attributes like "avalon role" and "direction"
# embedded in it.
#
# PORT_WIRING section: how it is.
55 #
# Now each port has its own PORT section, where the assignment names are
# its attributes, and the values are the values. Just like it should
# be.
#
60 # This function converts the old into the new, including parsing-apart
# the descriptor-string into useful data. When it's done, it deletes
# the old WIDTH and TYPE sections, to avoid confusion.
#

```

09880106-061201

```
#####
sub PTF_Update_Port_Wiring_Section
{
5   my ($db_Module) = (@_);
   my $changes = 0;

   my $db_Port_Type = &get_child_by_path($db_Module, "PORT_WIRING/TYPE");
   my $db_Port_Width = &get_child_by_path($db_Module, "PORT_WIRING/WIDTH");

10  $db_Port_Type or
    die ("PTF_Update_Port_Wiring_Section: no 'TYPE' section.");
  $db_Port_Width or
    die ("PTF_Update_Port_Wiring_Section: no 'TYPE' section.");

15  my $num_ports = &get_child_count ($db_Port_Type);
  for ($port_index = 0; $port_index < $num_ports; $port_index++)
  {
    my %port_hash = (); # put all PORT's assignments into here.

20    my $port_name      = &get_name (&get_child ($db_Port_Type, $port_index));
    $port_hash{width}   = &get_data_by_path ($db_Port_Width, $port_name);
    my $old_type        = &get_data_by_path ($db_Port_Type, $port_name);

    $old_type =~
25      /(master|internal|external)_(input|output|inout)?(shared)?(.*)/
    or die "Update: Port $port_name has malformed type: $old_type";

    $port_hash{direction} = $2;
    $port_hash{is_shared} = $3 ne "";
30    $port_hash{avalon_role} = $4;

    &delete_child($db_module, "PORT_WIRING/PORT $port_name");

    print "Adding PORT section: $port_name\n";
    foreach $assignment_name (keys(%port_hash)) {
      next if $port_hash{$assignment_name} eq "";

      my $value = $port_hash{$assignment_name};

40      &add_child_data
        ($db_Module, "PORT_WIRING/PORT $port_name/$assignment_name", $value);
    }
  }

45  &delete_child ($db_Module, "PORT_WIRING/TYPE");
  &delete_child ($db_Module, "PORT_WIRING/WIDTH");
}

50  #####
   #
   # Rename_Old_edf_File
   # If leonardo was run on nios 1.0, there's a $ptf.edf
   # lying around that will totally hose the user. 1.1 kit updates
55  # will not take effect in the synthesis design because quartus
   # will find the .edf file before it finds the wrapper. The
   # solution is to rename the edf file to something safe and warn
   # the user.

60  sub Rename_Old_edf_File
  {
    my ($ptf_filename) = @_;
```



```

my $malignent_edf_file = $ptf_filename;
$malignent_edf_file =~ s/\.ptf$/\.edf/;
my $benign_edf_file = "$malignent_edf_file\_old";
while (-e "$benign_edf_file"){ $benign_edf_file =
5      "$benign_edf_file_old";}
print STDERR "Renaming $malignent_edf_file to $benign_edf_file\n";
if (rename $malignent_edf_file,$benign_edf_file)
{
10   print STDERR "Renamed $malignent_edf_file to $benign_edf_file\n";
}
else
{
15   print STDERR "Rename Failed ($!).  Quartus will try to use\n";
   print STDERR "$malignent_edf_file instead of your design unless\n";
   print STDERR "it is specifically in the Quartus Project File list\n";
}
}

#####
20 # Set_Avalon_Defaults
#
# Takes the port list from every module in db_Sys and sets default
# width and direction for each port that does not have it defined.
# this function should not be called in 1.1
25
sub Set_Avalon_Defaults
{
   die "Set_Avalon_Defaults you should not call this function\n";
   #do not use this function
30   my ($db_Sys,
       $db_PTF_File) = @_ ;

   warn "in SAD\n";
   my $default_hash = &Get_Avalon_Requirement_Table;
35
   my @db_Module_Array = &get_first_children_of_type($db_Sys,
                                                       "MODULE");
   warn "dbma is @db_Module_Array\n";
   my $Mod;
   my %Sys_Hash;
40   my $Sys = \%Sys_Hash;

   foreach $db (@db_Module_Array)
   {
45       my $db_SBI = &PTF_Get_Required_Child_By_Path ($db,
                                                       "SYSTEM_BUILDER_INFO");
       $Mod = &PTF_Build_Hash_From_Section ($db_SBI);
       &PTF_Check_Bool ($Mod, "Is_Bus_Master",0);
       &PTF_Eval ($Mod, "Address_Width" );
50       &PTF_Eval ($Mod, "Data_Width" );

       $$Mod{name} = &get_data ($db);
       $Sys_Hash{master_data_width} = $$Mod{Data_Width};
       last if $Mod->{Is_Bus_Master};
55   }

   die "Set_Avalon_Defaults, No Master Found\n"
       unless ($$Mod{name});
   warn "SAD found master $$Mod{name}\n";
60

   #now we know which module is master
   #loop through again and get port data for each module
   my $default_direction;

```

09200106"061201

```

foreach $db_Mod (@db_Module_Array)
{
    my $module_name = &get_data($db_Mod);
    print STDERR "port $module_name\n";

    $default_direction = $default_hash->{required_slave_dir};
    $default_direction = $default_hash->{required_master_dir}
    if ($module_name eq $$Mod{name});

    my $db_Port_Wiring = &get_child_by_path($db_Mod,"PORT_WIRING",0) or die
        "Set_Avalon_Defaults, No PORT_WIRING SECTION FOUND FOR\n"
        . "MODULE $module_name\n";

    my @db_Ports = &get_first_children_of_type($db_Port_Wiring,
        "PORT") or die
        "Set_Avalon_Defaults, No PORTS FOUND in PORT_WIRING SECTION FOR\n"
        . "MODULE $module_name\n";

    foreach $db_Port (@db_Ports)
    {
        my $port_name = &get_data($db_Port);
        my $avalon_role = &get_data_by_path($db_Port,"avalon_role");
        if (!$avalon_role)
        {print STDERR " did not find avalon role for port $port_name\n";next}

        my $direction = &get_data_by_path($db_Port,"direction");
        if (!$direction)
        {
            $direction = $default_direction->{$avalon_role};
            print STDERR " did not find direction for port $port_name
($avalon_role)\n";
            print STDERR " going with $direction\n";
            my $test = &get_child_by_path($db_Port,"direction",1);
            print STDERR "so here is direction $test\n";
            &add_child_data($db_Port,"direction",$direction);
            my $after_test = &get_data_by_path($db_Port,"direction");
            print STDERR "after assigning direction, $after_test\n";
        }
        #or
        # $default_direction->{$avalon_role};

        my $W = &get_data_by_path($db_Port,"width");# or
        if (!$W)
        {
            eval ($default_hash->{width_default}{$avalon_role});
            print STDERR " did not find direction for port $port_name
($avalon_role)\n";
            print STDERR " going with $W\n";
            &add_child_data($db_Port,"width",$W);
            my $test = &get_child_by_path($db_Port,"width",1);
            print STDERR "so here is width $test\n";
            my $after_test = &get_data_by_path($db_Port,"width");
            print STDERR "after assigning width, $after_test\n";
        }

    }

    #&write_ptf_file ($db_PTF_File) or die
    # "ERROR Set_Avalon_Defaults, could not write ptf file ($!)\n";
}

1; # Modules must say "1"--mustn't they?

```

09880106-061201

09880105.061201

s dram_pbm_gen

```
#####
# pbm_and_system_modules.vpp
#
# This file contains a bunch of VPP-code which generates
# the systems' PBM module and top-level system-module.
#
# It builds the system as-specified by the PTF-file you
# indicate. You do so indicate
#
#     Generate_System_Logic
#
# This one happy Perl function is the top-level call for
# creating synthesizable Verilog which implements the
# "system" (e.g. Nios system) described by a single
# PTF-file.
#
# The one-and-only argument is, of course, the name
# of the PTF-file itself.
#
# The result is a bunch of Verilog (and other) files
# deposited in the users' Quartus project directory.
#
# Naturally, &Generate_System_Logic is a pretty complex
# Perl function which relies upon various dedicated Perl subroutines
# and utilities to do its many jobs. Those sub-functions are
# also defined in this module.
#
#####
use ptf_update;
use mk_bsfc;

#####
# Things to document:
#
# Philosophy, including overview of data-structure
#
# difference between "active" and "asserted"
#

#####
# Notes for my friends:
#
# Add to system-level WIZARD_SCRIPT_ARGUMENTS section:
#     "Principal_Tri_State_Data_Bus"
#
# Add to module-level SYSTEM_BUILDER_INFO sections:
#     "Uses_Registered_Select_Signal"
#     "Uses_Tri_State_Data_Bus"
#
# Master's SBI needs:
#     "Data_Width"
#     "Address_Width"
#
# Narrow accesses from non-dynamic tri-state peripherals are filled
# with "undefined." Sorry. That's life.

#####
# Funny Things to test:
```

```

#
# System with absolutely zero wait-states.
#
# Systems with -all- tri-state peripherals.
5 #

#####
## ERRORS TO CHECK:
10 #
# Funny chip-selects (base not a multiple of span, span not power of two).
#
# Warn about unknown assignments in PTF/SDF file
#
15 # Check assignments as much as possible when PTF/SDF is parsed,
# give line numbers.
#
# Check to be sure all "instances" have unique names.

20 #####
# numerically, a subroutine that enables us to sort by numeric order instead
# of alphabetic order.
#####
sub numerically {$a <=> $b;}

25 sub Find_Max
{
    my $max = "";
    foreach $val (@_)
    { $max = $val if $val > $max || $max eq ""; }
    return $max;
}

30 #####
# &Debug
#
# First arg turns message(s) on/off.
#
40 #####
sub Debug
{
    my ($doit, @messages) = (@_);

    return if !$doit;
    my $space = "";
    foreach $msg (@messages)
    {
        print STDERR "DEBUG: $space$msg\n";
        $space = "  ";
    }
}

50 #####
# PBM_Progress
55 #
# Our own private progress-printing routine.
# Uses the one in "wiz_utils.pm," except that it
# qualifies the output so that it only shows-up if the
# PBM_VERBOSE flag is on, and if this is pass 2.
60 #
#####
sub PBM_Progress
{

```

09380106"061201
TOTAL

FOR290" 50102200

```
my $old_out = select(STDOUT);
&Progress (@_);
select ($old_out);
}
5
#####
# PTF_Err
#
# In the future, I'd like to put a little more informative
10 # gingerbread around error messages--like the line number and such.
# This'd be the place to do it.
#
#####
sub PTF_Err
15 {
    my $msg;
    ($msg) = (@_);

    die ("Error while reading PTF file.\n
20         $msg");
}

#####
# Get_Port_By_Role
#
# Simple utility-function for acting on %Mod-hashes.
# You say what "avalon role" you want, and this returns
# a %Port-hash (by reference) for the corresponding module port.
#
30 # If there's more than one port with the avalon-role you asked for,
# then you get one of them at-random. This is only reliable if the
# avalon-role uniquely describes one of the module's ports.
#
# If you set the "$strict" option, you get an error if the port
35 # doesn't exist.
#
#####
sub Get_Port_By_Role
40 {
    my ($Mod, $avalon_role, $strict) = (@_);

    # Do step-by-step hash-dereferencing using temporary variables--
    # otherwise the syntax gets impenetrable.
    #
45 my $avalon_port_table = $$Mod{avalon_port_table};
my $Port = $$avalon_port_table{$avalon_role};

    die "Get_Port_By_Role: No port of type $avalon_role on module $$Mod{name}."
    if $strict && !$Port;

50     return $Port;
}

#####
55 # Get_Sys_Signal
#
# This is a utility-function that works on one of our
# defined-by-convention %Mod-hashes (which, you certainly remember,
# is a hash containing various useful information about each module)
#
60 # In particular, each module has a bunch of ports, some of which
# have an "avalon role." Any port with an "avalon role" connects
# to some ritualistically-named signal at the system level.
```

```

#
# Sometimes, given a module and an avalon role, it is useful to
# know what system-level signal "serves" that role.  As an example,
# it is useful to be able to answer the question:
5 #
# -- Which system-level signal drives the "address"-type port
# on the module "Uart_3?"
#
# This function here answers that very sort of question by
10 # digging the appropriate information out of the %Mod-hash you
# pass-in (by reference, of course).
#
# If you set the "$strict"-option argument, then we complain if
# no such port is found on the indicated module.  Otherwise, we
15 # return "" (null) for nonexistent ports.
#
#####
sub Get_Sys_Signal
20 {
    my $Port = &Get_Port_By_Role (@_);

    return "" if !$Port;    # Explicit null-string return if port doesn't exist.

    return $$Port(system_signal);
25 }

#####
# Get_Sys_Module_List
#
# This is a utility-function that works on one of our
30 # defined-by-convention %Sys-hashtables (which, you certainly remember,
# is a hash containing various useful information about the system-
# under-construction).
#
# This returns a list of %Mod-hash refs.  This list includes
35 # -all- modules in the system, including the master.
#
# The astute reader will note that a call to this function
# can be replaced by a single expression:
40 #
#     values %{$$Sys{module_table}}
#
# But that's one ugly expression.  This function adds a bit
# of self-documentation, and a much-needed dose of object-oriented,
45 # implementation-hiding, pseudo-access-method methodology.
#
#####
sub Get_Sys_Module_List
50 {
    my ($Sys) = (@_);

    my $module_table = $$Sys{module_table};    # Just for nice syntax.
    return values (%$module_table);
55 }

#####
# Get_Sys_Slave_List
#
# Just like "Get_Sys_Module_List," above, except that the list
60 # you get doesn't include the master.
#
# Many times, this is what you really wanted.  And this function
# saves you the trouble of having to put a master-exclusion test

```

00000106 061201

```

# in your otherwise- tidy loop.
#
#####
sub Get_Sys_Slave_List
5 {
    my ($Sys) = (@_);

    my @result = ();

10    foreach $Mod (&Get_Sys_Module_List($Sys))
    {
        push (@result, $Mod) unless $$Mod{Is_Bus_Master};
    }

15    return @result;
}

#####
# Get_Module_Port_List
20 #
# Just like "Get_Sys_Module_List," above-- and intended to
# serve the same dubious code-beautification purpose.
#
# The difference is: this function gets all the %Port-hashes
25 # out of a %Mod-hash, instead of all the %Mod-hashes out of a
# %Sys-hash.
#
#####
sub Get_Module_Port_List
30 {
    my ($Mod) = (@_);

    my $port_table = $$Mod{port_table}; # Just for nice syntax.
    return values (%$port_table);

35 }

#####
40 # Emit_Comment
#
# Emit the user-supplied string as a verilog comment directly
# into the output file. As a courtesy, we put //-characters
# at the beginning of every line, sparing the user the trouble.
45 #
# We call "&Vprint," so the user must previously have "selected"
# their destination verilog-file as STDOUT.
#
#####
50 sub Emit_Comment
{
    my ($comment, $language) = (@_);
    $language = "verilog" if !$language;

55    my $cstart = "-- ";
    $cstart = "// " if $language =~ /^verilog/i;

    $comment = $cstart.$comment;
    $comment =~ s|\n|\n$cstart|mg; # Put '// ' in front of every line.

60    &Vprint ("$comment\n");
}

```

09330106-061201


```
#####
# Emit_Top_Comment
#
# Writes module-name, date-stamp, and legal notice into the
5 # currently-selected output file, trilingually.
#
#####
sub Emit_Top_Comment
{
10   my ($module_name, $lang) = (@_);

   $lang = "verilog" if !$lang;

   my $date = scalar (localtime());
15   my $magic_altera_string = '%Altera Excalibur Nios(tm)%';
   my $stop_comment=<<EOM;
// megafunction wizard: $magic_altera_string
//// GENERATION: STANDARD
//// VERSION: WM1.0
20 // Module: $module_name
//
// Automatically-generated file: **** DO NOT EDIT ****
// Generated by Excalibur SOPC-Builder    [$date]
//
25 $GLOBAL_COPYRIGHT_NOTICE
//
EOM

   $stop_comment =~ s|//|--|mg if $lang =~ /^(vhdl|ahdl)/i;
30   &Vprint ($stop_comment);
}

#####
# Emit_Module_Header
#
# You give the name of the module ("Foo"), and this function
# emits the ritualistic top-of-module stuff, which we once would have
# done this way:
#
40 #   module Foo ( /*{&Declare_Ports_For("Foo")}*/ )
#               /*{ &Define_Ports_For("Foo")}*/
#
# Emits the module- and port-declarations for the named module into
# the currently-selected output file.
45 #
# Naturally, you have to have already done a &List_Ports_For the named
# module.
#
#####
50 sub Emit_Module_Header
{
   my ($module_name, $lang, $do_bb_declaration) = (@_);

   $lang = "verilog" if !$lang;

55
   if (($lang =~ /^vhdl/i)) {
       &Vprint ("LIBRARY ieee;\n");
       &Vprint ("use ieee.std_logic_1164.all;\n");
       &Vprint ("ENTITY $module_name IS\n");
       &Define_Ports_For ($module_name, 0, $lang);
       &Vprint ("END $module_name;\n");
       &Vprint ("ARCHITECTURE behavior OF $module_name IS\n");
60
   }
}
```

```

    } elsif ($lang =~ /^ahdl/i) {
        &Vprint ("SUBDESIGN $module_name\n");
        &Vprint ("(\n");
5      &Define_Ports_For ($module_name, 0, $lang);
        &Vprint (")\n");

    } elsif ($lang =~ /^verilog/i) {
        # Verilog:
10      my $synplify_bb_string = '/* synthesis syn_black_box */'
        if $do_bb_declaration;

        &Vprint ("module $module_name (\n");
        &Declare_Ports_For ($module_name);
15      &Vprint (") $synplify_bb_string ;\n");
        &Define_Ports_For ($module_name, 0, $lang);
        &Vprint ("\n // synopsys translate_off \n")
        if $do_bb_declaration;
        &Vprint ("\n");
20    } else {
        die "Emit_Module_Header: Foul language ($lang)";
    }
}

#####
# Emit_VHDL_Component
#
# You must declare all your VHDL black-boxes ("COMPONENT"s) in
# the "ARCHITECTURE" section of your module, before the first
# "BEGIN." OK, so be it.
#
# You must have previously done a &List_Ports_For -call for this
# module.
#
35 #####
sub Emit_VHDL_Component
{
    my ($comp_name) = (@_);
    &Vprint ("COMPONENT $comp_name IS\n");
    &Define_Ports_For ($comp_name, 0, "vhdl");
    &Vprint ("END COMPONENT;\n");
40 }

#####
# PBM_Assign
#
# Emit a simple assignment into the PBM-file (which we presume
# to be the currently-selected output file).
50 #
# We take special measures to avoid redundant assignments (we
# keep our own private hash of past assignments). We check to make
# sure the same signal is never assigned to two different things.
#
#
55 # The arguments are the target-signal and the value we want
# assigned to it.
#
# Also, note that we -explicitly- do nothing if the assignment-target
# is NULL (""). This deals gracefully with, for example, broadcast-
60 # assignments to modules that don't have a recipient port. For example,
# you can go ahead and &PBM_Assign the write-data bus to a module that
# doesn't actually have a "writedata"-type port, and it still works
# out OK (nothing happens).

```

```

#
#####
%__Private_PBM_Assign_Hash__;
sub PBM_Assign
5  {
    my ($target_signal, $assignment_value) = (@_);

    return if $target_signal eq "";    # Deal with null-assignment, per comment.

10    my $previous_assignment = $__Private_PBM_Assign_Hash__{$target_signal};

    # Ignore truly-redundant assignments:
    return if $previous_assignment eq $assignment_value;

15    $previous_assignment eq "" or die "
        Inconsistent assignments to signal $target_signal:
            ($previous_assignment) and ($assignment_value)";

    &Vprint ("assign $target_signal = $assignment_value;\n");

20    $__Private_PBM_Assign_Hash__{$target_signal} = $assignment_value;
}

#####
25 # PBM_Wire
#
# Emit a Verilog "wire" declaration into the currently-selected
# output file.  You give the name and width of the wire you
# want to declare.  Deals gracefully with the null wirename ("") and
30 # with zero-width wires:  No declaration is emitted.
#
# Also allows you to pass-in an optional "$assignment" Verilog-expression,
# so you can declare a wire and assign a value to it in one stroke.
#
35 #####
sub PBM_Wire
{
    my ($wirename, $width, $assignment) = (@_);

40    $width = 1 if $width eq "";

    return if $width == 0;
    return if !$wirename;

45    my $range = &W($width);

    &Vprint ("wire $range $wirename;\n") if $wirename && $width;

    &PBM_Assign ($wirename, $assignment) if $assignment;

50 }

#####
# Validate_And_Reserve_Address_Range
#
55 # Given a reference to a %Mod-hash, we compute the
# address-range allocated to that module and make sure that
# somebody else doesn't already live there.  If so, we
# print an error.
#
60 # Also, while we're thinking about the address, we do some sanity-checks.
# That's why the %Sys-hash (ref) is also passed-in, so we can
# check to be sure the module is in-bounds.
#

```

09860106.061201

09880106 "051201

```
# We also declare the silly "&within"-function for code-beauty.
#
#####
sub within { my ($test, $lo, $hi); return ($test >= $lo) && ($test <= $hi);}
5 %__Private_Address_Range_Hash__;

sub Validate_And_Reserve_Address_Range
{
10   my ($Mod, $Sys) = (@_);

   $$Mod{Base_Address} ne "" or die
       "Error: bad Base_Address setting for module $$Mod{name}";

   $$Mod{address_span} = 2*($$Mod{highest_address_bit_used} + 1);
15   $$Mod{end_address} = $$Mod{Base_Address} + $$Mod{address_span} - 1;

   $$Mod{end_address} <= ($$Sys{address_span} - 1) or die "
       End-address of module $$Mod{name} is greater than maximum system
       address ($$Sys{address_span} - 1)";

20   foreach $inst (keys(%address_range_list))
   {
       my ($min,$max) = split (/\/,,$__Private_Address_Range_Hash__{$inst});

25       die "Address-range conflict between $inst and $$Mod{name}.
           $inst occupies: [$min ...$max]
           $$Mod{name} occupies: [$$Mod{Base_Address} .. $Mod{end_address}]"
           if &within ($$Mod{Base_Address}, $min, $max) ||
               &within ($$Mod{end_address}, $min, $max) ;

30   }

   $__Private_Address_Range_Hash__{$$Mod{name}} =
       "$$Mod{Base_Address},$$Mod{end_address}";

35 }

#####
# Resolve_Address_Alignments
#
# In the PTF-file, the user can specify "dynamic" and "native"
40 # address-alignments. In these cases, the system-generator (that'd be
# me) needs to "do something smart" to reconcile peripherals with
# nonnative data-widths.
#
# We can only do that after all modules have been read-in (including,
45 # significantly, the master), and some system-level info has been
# set-up.
#
# Consequently, this function is called at the end of
# &Get_System_Data_From_PTF, after all the modules have been read.
50 #
#####
sub Resolve_Address_Alignments
{
55   my ($Sys) = (@_);

   foreach $Mod (&Get_Sys_Slave_List($Sys))
   {
       # Figure out what kind of address will ultimately be presented to this
       # module. This is slightly tricky because of the special "dynamic" case,
       # in which case we have to look at the data width.
60       #
       if ($$Mod{Address_Alignment} eq "dynamic")
       {
```

```

# Dynamic alignment: address-type depends on the data size:
$$Mod{address_type_used} =
    $$Mod{Data_Width} > 16 ? "word" :
    $$Mod{Data_Width} > 8 ? "halfword" :
    "byte" ;

} elsif ($$Mod{Address_Alignment} eq "native") {
    $$Mod{address_type_used} =
        $$Sys{master_data_width} == 16 ? "halfword" :
        "word" ;

} else {
    # "Normal" old-style alignment:
    warn ("
        Old-style Address_Alignment ($$Mod{Address_Alignment})
        found for module $$Mod{name}.\n") if $$Sys{verbose};
    $$Mod{address_type_used} = $$Mod{Address_Alignment};
}

# When is a "dynamic" module -not- "dynamic"? When it happens
# to be the same width as the master. Check width here, and set
# a per-module flag which tells us whether to really, truly use
# dynamic bus-sizing for this module.
#
if (($$Mod{Address_Alignment} eq "dynamic" ) &&
    ($$Mod{Data_Width} < $$Sys{master_data_width})) {
    {
        $$Mod{is_dynamically_sized} = 1;
    } else {
        $$Mod{is_dynamically_sized} = 0;
    }

# It's nice to know if the system has any dynamic bus-sizing:
$$Sys{has_dynamic_bus_sizing} = 1 if $$Mod{is_dynamically_sized};

# This is a handy thing to know about a module:
$$Mod{highest_address_bit_used} =
    $$Mod{address_type_used} eq "byte" ? ($$Mod{Address_Width} - 1) :
    $$Mod{address_type_used} eq "halfword" ? ($$Mod{Address_Width} ) :
    ($$Mod{Address_Width} + 1) ;
}

}

#####
# Get_System_Data_From_PTF
#
# Given a reference to a mostly-empty %Sys-hash, we read the
# PTF file, build a %Mod-hash for every module and a %Port-hash
# for every port, and stuff the results back into the %Sys
# data structure.
#
# We do as much "peephole" error-checking as we can while
# reading-in ports -- confirming allowed values for PTF fields,
# screening out "impossible" port types, etc.
#
# But there is a certain amount of checking that we -can't- do
# until we've read-in the entire system--checking to be sure no
# module's data bus is wider than the master, for example, has to
# wait until all modules are read-in. Those kinds of checks
# -do not- get executed here.
#

```

09860106"061201

```

# Plus, we don't actually "do anything" with the data. We do
# only pre-processing on the %Sys-hash, so that it will be
# easier, later on, to do what we need.
#
5 #####

my %PBM_HDL_EXTENSION;
$PBM_HDL_EXTENSION {verilog} = "v";
$PBM_HDL_EXTENSION {vhdl}    = "vhd";
10 $PBM_HDL_EXTENSION {ahdl}   = "tdf";

sub Get_System_Data_From_PTF
{
15   my ($Sys, $db_Sys) = (@_);

   # A hash of listrefs. The hash-keys are shared-port names:
   my %shared_port_table;

   #keys: shared port-names. Values: bus-group names.
20   my %bus_membership_table;
   my @tri_state_bus_list = ();

   # For Perl syntax-niceness, build these hashes up in the
   # module loop, then add them to the %Sys datastructure when
   # we're all done:
25   #
   my %module_table; undef %module_table;

   #####
   # Module loop
   #
   # Accumulate a hash of direct and derived information about each
   # module.
   # When we're all done, we'll add a reference to this hash to
30   # %Sys.
   #
   my $num_children = &get_child_count ($db_Sys);
   for ($child_index = 0; $child_index < $num_children; $child_index++)
   {
40     my $db_Module = &get_child ($db_Sys, $child_index);
     next if &get_name ($db_Module) ne "MODULE"; # ignore non-modules.

     #####
     # Read SYSTEM_BUILDER_INFO section
45     #
     # This will form the basis for our own private cache of
     # useful info about this module. Also, now would be a good
     # time to pre-digest and validate all the settings we read
     # from the PTF file. By this I mean: Converting "TRUE/FALSE"
50     # strings into testable bits, evaluating numerical expressions,
     # and checking for illegal values.
     #
     my $db_SBI = &PTF_Get_Required_Child_By_Path ($db_Module,
                                                    "SYSTEM_BUILDER_INFO");
55     my $sbi = &PTF_Build_Hash_From_Section ($db_SBI);
     my %Mod = %$sbi;
     $Mod{name} = &get_data ($db_Module);
     $Mod{class} = &PTF_Get_Required_Data_By_Path ($db_Module, "class");

60     &PBM_Progress ("Processing module $Mod{name}." ) if $$Sys{verbose};

     &PTF_Check_Bool (\%Mod, "Is_Enabled", 1);
     next unless $Mod{Is_Enabled}; # Quit early for disabled modules.

```

09830106.061201

09330106 "061201"

```
5  &PTF_Check_Bool (\%Mod, "Instantiate_In_System_Module", 1);
    &PTF_Check_Bool (\%Mod, "Uses_Registered_Select_Signal", 0);
    &PTF_Check_Bool (\%Mod, "Uses_Tri_State_Data_Bus", 0);
    &PTF_Check_Bool (\%Mod, "Has_IRQ", 0);
    &PTF_Check_Bool (\%Mod, "Is_Bus_Master", 0);
    &PTF_Eval (\%Mod, "Base_Address" );
    &PTF_Eval (\%Mod, "IRQ_Number", "N/A");
    &PTF_Eval (\%Mod, "Address_Width" );
10  &PTF_Eval (\%Mod, "Data_Width" );
    &PTF_Eval (\%Mod, "Read_Wait_States", "peripheral_controlled");
    &PTF_Eval (\%Mod, "Write_Wait_States", "peripheral_controlled");
    &PTF_Eval (\%Mod, "Setup_Time" );
    &PTF_Eval (\%Mod, "Hold_Time", "half_clock");
15  &PTF-Allow (\%Mod, "Address_Alignment", "dynamic", "native",
    "byte", "word", "halfword");

# Name module, if so far unnamed:
$Mod{Instance_Name} = "the_$Mod{name}"
20  if (($Mod{Instance_Name} eq "" ) ||
    ($Mod{Instance_Name} eq "--unknown--" ) );

$module_table{$Mod{name}} = \%Mod; # Put reference into system hash.

25  #####
    # If this is the master-module,
    # set a system-level variable, and check to see
    # that this is the only one.
    if ($Mod{Is_Bus_Master})
30  {
        $$Sys{master_name} eq "" or die "
        $$Sys{name} has multiple masters: $Mod{name} and $$Sys{master_name}";

        $$Sys{master_name} = $Mod{name};
        $$Sys{master} = \%Mod;
35  }

    $$Sys {has_registered_select_signals} = 1
        if $Mod{Uses_Registered_Select_Signal};
40  $$Sys {has_tri_state_data_busses} = 1
        if $Mod{Uses_Tri_State_Data_Bus};

45  #####
    # Port Loop
    #
    # Look at each port on this module. Build-up a hash
    # of useful information. For most "normal" ports, we
50  # can add a corresponding port on the PBM and/or the system.
    # Shared ports get recorded in a hash, so we can
    # add them to the system/PBM later, after all the port data
    # has been gathered for all modules.

55  if (&get_child_by_path($db_Module, "PORT_WIRING/TYPE")) {
        warn ("pbm_gen: old-style PORT_WIRING section for $Mod{name}.")
        if $$Sys{verbose};
        &PTF_Update_Port_Wiring_Section ($db_Module);
60  }

    my $db_Port_Wiring = &get_child_by_path($db_Module, "PORT_WIRING");
```

```

# Hashes which are included, by reference, in the %Module
# data structure. We build them up as temporary variables
# and stick them into the %Mod-hash at the end--otherwise,
# the Perl dereferencing syntax becomes impenetrable:
5  #
my %avalon_port_table = ();
my %port_table        = ();

my $num_ports = &get_child_count ($db_Port_Wiring);
10 for ($port_index = 0; $port_index < $num_ports; $port_index++)
{
    my $db_Port = &get_child ($db_Port_Wiring, $port_index);
    next unless &get_name($db_Port) eq "PORT";

15     # Whatever the PTF says about this port, we want to know.
    my $Port      = &PTF_Build_Hash_From_Section ($db_Port);
    $$Port{name}  = &get_data ($db_Port);
    my $who_died  = "$$Port{name} on module $Mod{name}"; # for errors.

20     # Test some basic stuff:
    $$Port{direction} =~ /^(input|output|inout)$/ or
        die "$who_died: Bad direction '$$Port{direction}'";

    # Port-record has pointer to module parent, and module
    # record has table of all Port-records:
    $$Port{parent}      = \%Mod;
    $port_table{$$Port{name}} = $Port;

25     &PBM_Progress (" Processing port $$Port{name}.") if $$Sys{verbose};

    # Ports -used to- have scopes. Now we can infer their scope
    # from the module's (and port's) other attributes.
    if ($$Port{scope}) {
        warn ("obsolete port '$$Port{name}' (has 'scope').\n")
30         if $$Sys{verbose};

        $$Port{scope} =~ /^(internal|external|master)$/ or
            die "$who_died: Bad scope '$$Port{scope}'";

40         $$Port{is_external} = $$Port{scope} eq "external"; # Testable bool.
    }

    # Decide whether this port is esxternal or internal. We used
    # to require that the user tell us, but now we can figure it
    # out for ourselves:
45     #
    if (($Mod{Instantiate_In_System_Module})) {
        # For modules -inside- the system-module, all their
        # avalon-ports are internal. All their non-avalon ports
50         # are external
        $$Port{is_external} = 1 if !$$Port{avalon_role};
    } else {
        # For modules -outside- the system-module, all their
        # avalon-ports are external. All their non-avalon ports
55         # are none of our business
        $$Port{is_external} = 1 if $$Port{avalon_role};
    }

60     # Some consistency-checking:
    # * Only external signals can be "shared":
    # * All internal (or master) signals must have an avalon-role.
    die "$who_died: only external ports may be shared."

```

09880106-061201


```
if ( $$Port{is_shared} && !$$Port{is_external} );
```

```
#####
```

```
# What does this port connect to?
```

```
#
```

```
# For a "typical" system-internal module with nothing "shared,"  
# each port gets connected to a unique, dedicated wire in the  
# system module. That wire will go to one of two places:  
# a like-named port on the PBM (for avalon signals) or be  
# promoted to a system-level port. Easy enough.
```

```
# There are two wrinkles to consider: modules which are -not-  
# instantiated in the system, and modules which have shared  
# ports.
```

```
# *** External Modules
```

```
# If the module is not instantiated inside the system, then  
# we do twp special things:
```

```
# 1) Ignore any signals which don't have an avalon_role,  
# because they're not our responsibility, anyhow.
```

```
# 2) Promote its PBM-ports to system-level I/Os with the  
# same direction and name.
```

```
# **** Shared Ports
```

```
# Ports are only "shared" if they're part of a tri-state  
# bus structure. All tri-state busses are named. Here  
# are some examples of system-level signals which are part  
# of a shared tri-state bus:
```

```
#         memory_bus_address  
#         memory_bus_byteenablen
```

```
#         ide_data  
#         ide_writen
```

```
# In this case, the signal names are a concatenation of the  
# tri-state bus group name and the avalon role. We don't  
# add these ports to the PBM/system -yet-, because we don't  
# really know their widths until all the modules have been  
# processed. Instead, we just record our %Port as a "client"  
# of this shared port in a hash. Later, we'll work out  
# exactly how the shared ports show up on the system module.
```

```
# if ($$Port{is_shared})
```

```
{
```

```
    die "  
        Shared port $$Port{name} on module $$Mod{name} has no  
        'Avalon role'"  
        if (!$$Port{avalon_role});
```

```
    die "  
        Shared port $$Port{name} found on module $Mod{name}, but  
        module does not use tri-state data bus"  
        if ((!$Mod{Uses_Tri_State_Data_Bus} ) ||  
            ( $Mod{Tri_State_Data_Bus} eq "" ) );
```

```
    my $shared_port = "$Mod{Tri_State_Data_Bus}_$$Port{avalon_role}";  
    $$Port{system_signal} = $shared_port;
```

09880106-061201

```

# Record the fact that this %Port is a client of
# this shared port.  Push a reference to this %Port-hash
# onto this shared-port client list:
#
5   push (@{$shared_port_table{$shared_port}}, $Port);

# also record which tri-state bus group this shared port
# belongs to.  (true, you could figure it out by looking
# at it's name, but that just sounds risky to me:
10  #
    $bus_membership_table{$shared_port} = $Mod{Tri_State_Data_Bus};
} else {

    # This is -not- a shared port, so we make one of those
    # much-beloved machine-generated port names
    # (e.g. bidir_port_to_and_from_the_lcd_pio).
    #
15  my $transfer_str = ($$Port{direction} eq "input") ? "to" :
                      ($$Port{direction} eq "output") ? "from" :
                      "to_and_from";

20  $$Port{system_signal} =
    "$$Port{name}_$transfer_str\_$_Mod{Instance_Name}";
}

25  # Construct an inverse hash so we can look-up ports
    # -by avalon role- for this module:
    #
    $avalon_port_table {$$Port{avalon_role}} = $Port
        if $$Port{avalon_role};

30  #####
    # Record some useful system-level and
    # module-level information as the ports
    # go by:
    #
    $$Sys{master_address_width} = $$Port{width}
        if ($Mod{Is_Bus_Master} ) &&
        ($$Port{avalon_role} eq "address" ) ;

40  $$Sys{master_data_width} = $$Port{width}
        if ($Mod{Is_Bus_Master} ) &&
        ($$Port{avalon_role} eq "writedata" ) ;

45  } #end: %Port-loop

# Attach the hashes we built-up in the %Port-loop into the
# %Mod data structure:
#
50  $Mod{port_table} = \%port_table;
    $Mod{avalon_port_table} = \%avalon_port_table;

#####
# Derived Module Info
#
55  # Pre-digest some useful facts about this module:
    #
    # It's nice to have a list of all tri-state busses in the system:
    push (@tri_state_bus_list, $Mod{Tri_State_Data_Bus})
60  if $Mod{Uses_Tri_State_Data_Bus};

# Predigest hold-time values: handle "half-clock" case.
$Mod{hold_time_full_clocks} = $Mod{Hold_Time};

```

09880106 "061201

$$\text{\$Mod}\{\text{hold_time_full_clocks}\} = 0 \text{ if } \text{\$Mod}\{\text{Hold_Time}\} \sim \text{/half/;}$$

```
} # End: %Mod-loop
```

```

$$$Sys{module_table}      = \%module_table;
$$$Sys{shared_port_table} = \%shared_port_table;
$$$Sys{bus_membership_table} = \%bus_membership_table;
$$$Sys{tri_state_bus_list} = \@tri_state_bus_list;

```

#####

```
# Derived system-info
```

#

```

$$Sys{pbm_name}      = $$Sys{name} . "_pbm";
$$Sys{core_name}     = $$Sys{name} . "_core";

```

```

$$Sys{pbm_file}      = "$$Sys{system_directory}/$$Sys{pbm_name}.v";
$$Sys{core_file}     = "$$Sys{system_directory}/$$Sys{core_name}.v";

```

```
# Wrapper-file needs correct extension for target language.
```

```

--
$$Sys{hdl_language} = lc ($$Sys{hdl_language});

```

```
my $extension = $PBM_HDL_EXTENSION{$$Sys{hdl_language}};
```

```
$extension or die "Unrecognized 'hdl_language': $$Sys{hdl_language}";
```

```

    $$Sys{wrapper_file} = "$$Sys{system_directory}/$$Sys{name}.$extension";

```

```
# Only used if AHDL:
```

```
&Resolve_Address_Alignments ($Sys);
```

```
# ... might want to put code here to figure out
```

```
#      "principal" tri-state bus...
```

```
# This will probably be a call to some function that has
```

```
# a bunch of heuristics.
```

}

#####

```
# Get_Avalon_Requirement_Table
```

#

```
# Returns a handy table that we use as a verification
```

```
# template.  Then check each avalon-role port to be sure
```

```
# it agrees with what we expect.  The table should be evalled in
```

```
# a function that has $Sys and $Mod set accordingly
```

```
# (Isn't error checking a big pain?...Code was so much
```

```
# simpler in the Wild West):
```

#

```
# It's easier to check errors if you set defaults.
```

```
# That way you don't punish the user for supplying
```

```
# data you know already. I've added Width Default to the table.
```

Sorry Tim, you'll have to maximize your screen to see all the

三

```
sub Get_Avalon_Requirement_Table
```

```
my $requirement_table =
# Avalon Role      | Slave | Master | Width Requirement
```

Width Default

00000106"061201

```

"clk          | input  | input  | \SW == 1
\SW = 1,
resetsn       | input  | input  | \SW == 1
5  \SW = 1,
always0       | input  | input  | "\"Any width OK\"
\SW = "\"\",
always1       | input  | input  | "\"Any width OK\"
\SW = "\"\",
10 writen      | input  | output | \SW == 1
\SW = 1,
readn         | input  | output | \SW == 1
\SW = 1,
byteenablen   | input  | output | \SW <= 4
15 \SW = 4,
waitrequest   | output | input  | \SW == 1
\SW = 1,
irq           | output | input  | \SW == 1
\SW = 1,
irqnumber     | N/A    | input  | \SW <= 6
20 \SW = 6,
chipselct     | input  | N/A    | \SW == 1
\SW = 1,
registeredselectn | input  | N/A    | \SW == 1
\SW = 1,
ifetch        | input  | output | \SW == 1
25 \SW = 1,
memis32bits   | N/A    | input  | \SW == 1
\SW = 1,
data          | inout  | N/A    | \SW <= \\\$Sys{master_data_width}
30 \SW = \\\$Sys{master_data_width},

readdata      | output | input  | \SW == \\\$Mod{Data_Width} &&
\SW <= \\\$Sys{master_data_width}
\SW = \\\$Mod{Data_Width},
35 writedata   | input  | output | \SW == \\\$Mod{Data_Width} &&
\SW <= \\\$Sys{master_data_width}
\SW = \\\$Mod{Data_Width},
40 address     | input  | output | \SW > 0 &&
\SW <= \\\$Sys{master_address_width}
\SW = \\\$Mod{Address_Width},
";
# Turn our pretty text-table into a bunch of code-useful hashes:
45 #
#now some hashes;
my %required_slave_dir;
my %required_master_dir;
50 my %width_requirement;
my %width_default;

$requirement_table =~ s/\s+/ /sg;
foreach $req (split (/\\s*\\,\\s*/s, $requirement_table))
{
55     my ($role, $slave_dir, $master_dir, $width_condition,
$width_default_string) =
        split (/\\s*\\|\\s*/s, $req);

    $required_slave_dir {$role} = $slave_dir unless $slave_dir eq "N/A";
60     $required_master_dir {$role} = $master_dir unless $master_dir eq "N/A";
    $width_requirement {$role} = $width_condition;
    $width_default {$role} = $width_default_string;
}

```

```

my %return_hash;
$return_hash{required_slave_dir} = \%required_slave_dir;
$return_hash{required_master_dir} = \%required_master_dir;
5  $return_hash{width_requirement} = \%width_requirement;
$return_hash{width_default} = \%width_default;

return (\%return_hash);
}

10 #####
# Check_Avalon_Rules
#
# Given a (the?) system-hash, this function loops over the
15 # data structure module-by-module and port-by-port and checks
# that no system-level Avalon rules have been violated.
#
# These are rules like: All "chipselect"-type ports on a module must
# be 1 bit wide, all "address"-ports must be inputs, and all
20 # data ports must be narrower than the masters'.
#
# This function replaces the section of inline-code formerly known
# as "The Parade of the Port Types."
#
25 #####
sub Check_Avalon_Rules
{
    my ($Sys) = (@_);

30    my $requirement_hash_table = &Get_Avalon_Requirement_Table;

    my %required_slave_dir = %{$requirement_hash_table->{required_slave_dir}};
    my %required_master_dir = %{$requirement_hash_table->{required_master_dir}};
    my %width_requirement = %{$requirement_hash_table->{width_requirement}};
35    my %width_default = %{$requirement_hash_table->{width_default}};

    # Now consider each avalon-port on each module
    # and see if it lives up to our expectations.
    # If port is not defined, set it to default value.

40    foreach $Mod (&Get_Sys_Module_List($Sys))
    {
        my $avalon_port_table = $$Mod{avalon_port_table};
        foreach $role (keys (%$avalon_port_table))
        {
45            {
                my $Port = $$avalon_port_table{$role};
                my $W = $$Port{width};

                my $required_dir = $$Mod{Is_Bus_Master} ? \%required_master_dir :
50                \%required_slave_dir ;

                #set default width
                eval ($width_default{$role})
                    if ($W eq "");

55                #set default direction
                $$Port{direction} = $$required_dir{$role}
                if ($$Port{direction} eq "");

60                die "
                    Illegal avalon-role : $role
                    for port $$Port{name} on module $$Mod{name}."
                    if (!$$required_dir{$role});
            }
        }
    }
}

```

```

die "
  Illegal direction: $$Port{direction}.
  Expected: $$required_dir($role)
  for port $$Port{name} on module $$Mod{name}."
  if ($$Port{direction} ne $$required_dir($role));

my $width_ok = eval ($width_requirement($role));
die "Bad check-expression: $width_requirement($role) ($@)" if $@;

die "
  Illegal width: $W
  for port $$Port{name} on module $$Mod{name}."
  if (!$width_ok);

#####
# Per-port Consistency-checks
#
# Does this port agree with things that were said in the
# module's SBI section?
#
if ($role =~ /data$/) {
  $$Mod{Data_Width} == $$Port {width} or die "
    Width of data port $$Port{name} ($$Port{width}) is inconsistent
    with 'Data_Width' setting for module $$Mod{name}.";
}

if ($role eq "address") {
  $$Mod{Address_Width} == $$Port {width} or die "
    Width of address port $$Port{name} ($$Port{width}) is
    inconsistent with 'Address_Width' setting for
    module $$Mod{name}."
}
}

# Half-clock hold-time only allowed with zero setup-time:
if (($$Mod{Hold_Time} =~ /half/)) {
  $$Mod{Setup_Time} == 0 or die "
    Error in module $$Mod{name}: half-clock 'Hold_Time' setting
    allowed only if 'Setup_Time' is 0 (zero).";
}

next if $$Mod{Is_Bus_Master}; # The following checks are for mere slaves.

#####
# Funny Setup/Hold Rule
#
# Here's the rule:
#
# -- If you have a nonzero hold-time, then you get a nonzero
#    setup-time, whether you asked for one or not.
#
# Here's the explanation:
#
# You may recall that the masters' readn- and writen- signals come
# directly from the Q-output of registers. Timing-wise, this is
# a good and happy thing. Peripherals which don't explicitly request
# setup/hold times get the masters' registered strobe signals, and
# life is good.
#
# But now consider the plight of the poor module who requests
# setup/hold time. Necessarily, he now gets his own customized

```

09880106 061201 102190

```

# (narrower) version of the read- and write-strobes.  It would be
# oh-so-nice if his custom strobes also came directly from Q-outputs
# of registers.  (And, I note, it would also make the
# strobe-customization logic easier for your humble implementor).
5 # So now we have the custom readn- and writen-strobes coming from
# register outputs.  But, obviously, this happy register introduces
# a one-clock delay between the time we decide to assert the custom
# strobe and the time it gets asserted.  That's fine--we have
10 # (at least) one clock-cycle of slack before we need to assert the
# strobe--as long as there's a (nonzero) setup-time.  That gives
# us the clock we need to "customize" and still have a register.
# So.  Phrased concisely:
#
# Custom strobes want registers
15 # registers imply delay
# nonzero setup-time accomodates delay
#
# --> custom strobes need a nonzero setup-time.
#
20 # We print a warning when we encounter such a case.
#
if (($Mod{Setup_Time} == 0) &&
    ($Mod{hold_time_full_clocks} > 0)
    )
25 {
    $Mod{Setup_Time} = 1;
    warn ("
        Setup-time (1 clock) automatically added for module $Mod{name}.
        Modules with nonzero hold-time automatically get at least
30         one clock of setup-time.\n
        ");
    }

# Some extra module-level consistency-checking between
# "System Builder Info" and port-types.
#

&Validate_And_Reserve_Address_Range ($Mod, $Sys);

40 die "Module $Mod{name} 'Has_IRQ', but no pin is of type 'irq'."
    if $Mod{Has_IRQ} && !$Savalon_port_table{irq};

die "Module $Mod{name} has an irq-pin, but 'Has_IRQ' is FALSE."
    if !$Mod{Has_IRQ} && $Savalon_port_table{irq};

45 die "Module $Mod{name} has word-alignment, but master is not 32 bits."
    if ($Mod{address_type_used} eq "word" &&
        $Sys{master_data_width} < 32
        );

50 die "Module $Mod{name}: Data is too wide for dynamic alignment."
    if ($Mod{is_dynamically_sized} &&
        $Mod{Data_Width} > ($Sys{master_data_width} / 2) );

die "Module $Mod{name}: must set SBI/Uses_Registered_Select_Signal."
55 if (!$Mod{Uses_Registered_Select_Signal} &&
    $Savalon_port_table{registeredselectn});

die "Module $Mod{name}: peripheral controlled wait \n".
    "not supported for registered chip selects\n"
60 if ($Mod{Uses_Registered_Select_Signal} &&
    (($Mod{Read_Wait_States} == /peripheral_controlled/i) ||
    ($Mod{Write_Wait_States} == /peripheral_controlled/i))
    );

```

```

#hold time means theres a setup time,
die "Module $$Mod{name}: peripheral controlled wait not\n".
    "supported for peripherals with non-zero setup and/or hold times\n"
5     if (($Mod{Setup_Time}) &&
        (($Mod{Read_Wait_States} =~ /peripheral_controlled/i) ||
        ($Mod{Write_Wait_States} =~ /peripheral_controlled/i))
        );

10    die "Module $$Mod{name}: can't find 'registeredselectn'-type port."
        if ($$Mod{Uses_Registered_Select_Signal} &&
            !$savalon_port_table{registeredselectn});

    my $Port_writen = $$savalon_port_table{writen};
15    die "Module $$Mod{name}: shared writen port illegal if setup/hold > 0."
        if ($$Port_writen{is_shared} &&
            ($$Mod{Setup_Time} + $$Mod{hold_time_full_clocks} > 0) );

    my $Port_readn = $$savalon_port_table{readn};
20    die "Module $$Mod{name}: shared readn port illegal if setup/hold > 0."
        if ($$Port_readn{is_shared} &&
            ($$Mod{Setup_Time} + $$Mod{hold_time_full_clocks} > 0) );

    # ... Add more, please...
25    # Check:
    # Uses_Registered_Select_Signal vs. actual "registeredselectn" ports.
    # Uses_Tri_State_Data_Bus vs. actual shared/data ports.
    # if principal bus declared, system -should- have tri-state busses.
    # principal tri-state bus must be as wide as CPU.
30    # if we have principal bus, at least one module must be on it.
    # You may -not- have shared readn/writen signals with nonzero
    # setup/hold times.
    # system data width should be only -exactly- 16 or 32.

35    } # End: $Mod-loop
}

#####
# Create_System_Port_Lists
40 #
# Given a reference to a (the?) system-hash, this function
# builds a "List_Ports_For-" definition of the system-module
# and the PBM.
#
45 # For the most part, this is easy: We look through the list of all ports
# on all modules. If any have an avalon-role, then their complement
# shows up on the PBM. If any are "external", then they show up on
# the system module.
#
50 # The one nasty little wrinkle is shared-ports. We have kept a separate
# record of all shared ports, and now we use it to compute the appropriate
# width--then we can add the shared ports to both the PBM and system modules.
#
# And then shared ports have one more trick: The width of shared address
55 # ports. For most shared ports, the width is just the width of the largest
# client. For address ports, we need to take -alignment- into account.
#
#####
sub Create_System_Port_Lists
60 {
    my ($Sys) = (@_);

    # The clk and reset_n ports are magic. They -always- appear

```

09330106:061201


```

# on both the system and the PBM.
#
$$Sys{pbm_list_ports_for_string} = "clk      | 1 | input,
                                     reset_n | 1 | input,";
5  $$Sys{system_list_ports_for_string} = $$Sys{pbm_list_ports_for_string};

foreach $Mod (&Get_Sys_Module_List($Sys)) {
  foreach $Port (&Get_Module_Port_List ($Mod))
  {
10    next if $$Port{is_shared};    # Do shared ports later.

    if ($$Port{avalon_role})
    {
15      my $pbm_port_dir = $Complementary_Direction ($$Port{direction});
      my $pbm_port_description =
        "$$Port{system_signal} | $$Port{width} | $pbm_port_dir,";

      $$Sys{pbm_list_ports_for_string} .= $pbm_port_description;
      $$Sys{system_list_ports_for_string} .= $pbm_port_description
20      if (!$$Mod{Instantiate_In_System_Module});
    } else {

      # OK, this port doesn't have an avalon role.  For
      # externally-instantiated modules, we just ignore it.
      # for internally-instantiated modules, we promote it
      # to a top-level system port with the -same- direction
      # as the port on the module.
      #
      # All non-avalon-type ports must be external
30      $$Sys{system_list_ports_for_string} .=
        "$$Port{system_signal} | $$Port{width} | $$Port{direction},"
        if ($$Mod{Instantiate_In_System_Module});
    }
  } # End: $Port-loop
} # End: $Mod-loop

#####
# Now do the shared ports.
#
40 my $shared_port_table = $$Sys{shared_port_table};

foreach $shared_port_name (keys(%$shared_port_table))
{
45   my $client_list = $$shared_port_table{$shared_port_name};

   # Look at all the clients.  Build-up enough information
   # to calculate the port-width (below).  Also, check that
   # all the clients have a consistent direction, avalon-role, etc.
   #
50   my $client_dir = "";
   my $shared_port_role = "";
   my $shared_port_width = 0;
   foreach $client_port (@$client_list)
   {
55     $shared_port_role = $$client_port{avalon_role} if !$shared_port_role;
     $client_dir = $$client_port{direction} if !$client_dir;

     $client_dir eq $$client_port{direction} or die "
       Shared port $shared_port_name has client $$client_port{name}
60       (direction is $$client_port{direction}; expected $client_dir).";

     $shared_port_role eq $$client_port{avalon_role} or die "
       Shared port $shared_port_name (role: $shared_port_role) has

```

```

        client $$client_port{name} (role: $$client_port{avalon_role}).";

#####
# Computing the width is tricky, but only for address ports.
# For everybody else, it's just the width of the widest client.
# For address ports, we have to take into account the fact that
# -all- shared address ports present the full byte-address,
# all the way down to A[0] (even if none of the clients use
# A[0]). Thus, the width of the shared port might be greater
# than the width of any (all) of the client address ports.
#
if ($shared_port_role eq "address")
{
    # Address port. Now we must inquire about its ancestry:
    my $parent_module = $$client_port{parent};
    my $net_width = $$parent_module{highest_address_bit_used} + 1;

    $shared_port_width = $net_width
        if $net_width > $shared_port_width;
} else {
    # Normal, non-address port: Width is just the max of clients:
    #
    $shared_port_width = $$client_port{width}
        if $$client_port{width} > $shared_port_width;
}
} # End: Client-port loop.

#####
# Shared data-bus ports.
#
# There's one weird circumstance where a shared-port is actually
# wider than the max of its clients: When it's the main
# tri-state data bus to the CPU. In this one case, the bus
# must be at least as wide as the CPU
#
if (($$Sys{Principal_Tri_State_Data_Bus}) &&
    ($shared_port_role eq "data" ) &&
    ($shared_port_name =~ /^$$Sys{Principal_Tri_State_Data_Bus}/)) {
    $shared_port_width = $$Sys{master_data_width};
}

# Shared ports always have an avalon-role, so they always show up
# on the PBM as the -complimentary- match for the client-port.
# also, shared ports are always external, so they are always promoted
# to like-named system-level ports.
#
my $shared_port_dir = $Complementary_Direction{$client_dir};
my $shared_port_description =
    "$shared_port_name | $shared_port_width | $shared_port_dir,";

$$Sys{pbm_list_ports_for_string} .= $shared_port_description;
$$Sys{system_list_ports_for_string} .= $shared_port_description;
}

# We've built those nice list-ports-for string, so let's use 'em:
#
&List_Ports_For ($$Sys{name}, $$Sys{system_list_ports_for_string});
&List_Ports_For ($$Sys{core_name}, $$Sys{system_list_ports_for_string});
&List_Ports_For ($$Sys{pbm_name}, $$Sys{pbm_list_ports_for_string});
}

#####

```

```

# Core_Emit_Wire_Declarations
#
# The system's "core" module is just a bunch of instances all wired-up
# together. Some "wires" in the core-module are actually external
5 # ports, so they're already declared in the cores' module
# declaration. Other "wires" are actual Verilog wires which we
# need to declare. They are the internal (module-to-module) signals
# which are never seen from the outside.
#
10 # This function emits wire-delcarations for all the internal signals.
#
# It's easy to identify all the internal signals: there's one for
# every module-port scoped "internal" or "master" -- in other words,
# there's a core-level wire for every non-external port on every
15 # module in the system. That makes it pretty easy:
#
#####
sub Core_Emit_Wire_Declarations
(
20     my ($Sys) = (@_);

    foreach $Mod (&Get_Sys_Module_List($Sys)) {
        foreach $Port (&Get_Module_Port_List ($Mod))
        {
25             next if $$Port{is_external};
            my $range = &W($$Port{width});
            &Vprint ("wire $range $$Port{system_signal};\n");
        }
30     }

    #####
    # Core_Emit_Instances
    #
35 # The system's core-module is a bunch of instances all wired-up
# together. This function emits the instantiation-statements
# for all system-modules, including the PBM.
#
# In general, any given port on a module does not necessarily
40 # connect to a core-level signal (wire) of that same name.
# The correspondence of what-signal-goes-to-what-port, though,
# is built-in to the %Sys-database hash. We extract this
# information and use it to instantiate each module in the
# system.
45 #
#####
sub Core_Emit_Instances
(
50     my ($Sys) = (@_);

    #####
    # Start with an instantiation of the PBM.
    #
55 # Note that the PBM doesn't appear on the system's {module_table}.
# Note also that all the ports on the PBM -do- connect to like-named
# signals at the core-level. This makes instantiation a snap:

    &Instantiate_And_Connect ($$Sys{pbm_name}, "the_$$Sys{pbm_name}");

60     #####
    # Now instantiate all the "regular" modules. But, before we do,
    # go through their ports and build up a correspondence ("exception")
    # table:

```

```

#
foreach $Mod (&Get_Sys_Module_List($Sys))
{
    # Skip external modules, of course.
5    next unless $$Mod{Instantiate_In_System_Module};

    my $list_ports_for_string = "";
    my %except;
    undef %except; # Superstition. I have no idea if this is called-for.
10
    foreach $Port (&Get_Module_Port_List ($Mod))
    {
        $except{$$Port{name}} = $$Port{system_signal};
        $list_ports_for_string .= "
15        $$Port{name} | $$Port{width} | $$Port{direction},";
    }

    &List_Ports_For ($$Mod{name}, $list_ports_for_string);
    &Instantiate_And_Connect ($$Mod{name}, "the_$$Mod{name}", "", \%except);
20 }
}

#####
# PBM_Emit_Dynamic_Bus_Sizer
#
# The dynamic bus-sizer makes any narrow peripheral "look like"
# it's full-width memory. For example, if you connect an 8-bit
# memory device to a 32-bit master using DBS, the CPU will "see"
# a full 32-bit value every time it does a LD- from the device.
#
# This is accomplished, of course, through a bunch of sequential
# logic (the DBS), which (in this example) fetches four successive
# bytes from the memory, assembles them into a full-width word,
# and presents it to the CPU (all the while the CPU has been held
# in a wait-state, of course).
#
# A similar thing happens during write-operations. When
# writing a 32-bit value to 8-bit memory (for example), the CPU will
# wait while we execute four successive write-operations to the
# memory. The DBS-logic is sensitive to the fact that sometimes
# the CPU will execute a narrow-write (e.g. a byte-write). In these
# cases, the DBS-logic is smart enough to execute only one
# write-cycle.
#
45 # Note that all of our "successive read operations" or "successive
# write operations" are subject to the bus-timing (wait states,
# setup/hold-times, etc.) for the target peripheral. In other words,
# each sub-operation of the DBS-unit is a full-fledged bus transaction
# controlled by the wait-state generator. You could, then, think
50 # of the DBS-operation being "laid on top of" the regular wait-state
# logic.
#
# FYI, dynamic bus sizing is very handy for memory-type devices,
# but doesn't make much sense for register-controlled peripherals
55 # (e.g. UARTs).
#
# This function here creates the dynamic bus-sizing logic.
# Note that there are, really, two independent DBS-units--one
# of which handles byte-wide peripherals, the other handles
60 # halfword-wide peripherals.
#
# KNOWN LIMITATION:
#

```



```

# We need to answer two questions:
#
# 1) When do we start a DBS-operation?
# 2) When do we "advance" to the next sub-operation (chunk)?
#
# Well, we start a new operation (1) when:
#
# 1a) The active peripheral needs dynamic bus-sizing.
#
# 1b) It's the start of a new bus-transaction
#      (indicated by the "bus_transaction_start" signal,
#      which is computed by the wait-state generator.
#
# 1c) There's not some other special, weird circumstance--like
#      narrow writes or "ifetch" going on which preclude the
#      need for a dynamic operation.
#
# And we advance to the next chunk (2) at the end of each
# bus cycle. Happily, the wait-state generator also indicates
# this by computing the signal "bus_cycle_end".
#
#
&PBM_Wire ("bus_cycle_end");           # Declared here, computed later.
&PBM_Wire ("bus_transaction_start");    # Declared here, computed later.

my $be_bus = &Get_Sys_Signal($$Sys{master}, "byteenablen");
&PBM_Wire ("write_is_narrow", 1,
           "| " . $be_bus);

if ($$Sys{master_data_width} == 32) {
    &PBM_Wire ("write_is_narrow_16", 1,
              "write_is_narrow && (($be_bus\[3\] == $be_bus\[2\]) &&
                ($be_bus\[1\] == $be_bus\[0\]) )");

    &PBM_Wire ("dbs_8_half_start", 1, "
              dbs_8_active           &&
              bus_transaction_start  &&
              (write_is_narrow_16)   ");
}

&PBM_Wire ("dbs_8_full_start", 1, "
              dbs_8_active           &&
              bus_transaction_start  &&
              (~write_is_narrow)     ");

my $ifetch_signal = &Get_Sys_Signal ($$Sys{master}, ifetch);
&PBM_Wire ("dbs_16_start", 1, "
              dbs_16_active          &&
              bus_transaction_start  &&
              (~$ifetch_signal)      &&
              (~write_is_narrow)     ");

#####
# Sequencing registers
#
# Here are the signals we must come up with:
#
#   dbs_8_byte_{0,1,2,3}
#   dbs_16_halfword_{0,1}
#
#   -- and--
#
#   dbs_8_write_byte_{0,1,2,3} ("0" not actually produced)

```

```

# db_16_write_halfword(0,1) ("0" not actually produced)
# (I think, in the absence of narrow writes, these are just
# equivalent to the non-write versions).
#
# db_wait_asserted
#
my @db_wait_asserted_terms = ("1'b0"); # "1'b0" fixes empty list.

if (scalar(@db_8_modules))
{
# How many additional bus cycles? 3 or 1, for 32- and 16-bit masters.
# These cases are similar-enough that it's tempting to make
# one piece of code that builds both, but I'm just going to break
# them into separate cases for clarity:
#
if ($$Sys{master_data_width} == 32)
{
&PBM_Wire ("db_8_byte_3");
&PBM_Wire ("db_8_byte_1");

&Delay ("out          = db_8_state_byte_3,
        sync_set      = db_8_full_start,
        sync_reset    = bus_cycle_end,
        reset         = ,
        ");

&Delay ("out          = db_8_byte_2,
        in            = db_8_state_byte_3,
        enable        = bus_cycle_end,
        reset         = ,
        ");

# Two versions of byte-1 signal, because it can be set from
# two different sources: The continuation of a 4-byte
# operation, or the start of a two-byte operation.
#
&Delay ("out          = db_8_continue_byte_1,
        in            = db_8_byte_2,
        enable        = bus_cycle_end,
        reset         = ,
        ");

&Delay ("out          = db_8_state_byte_1,
        sync_set      = db_8_half_start,
        sync_reset    = bus_cycle_end,
        reset         = ,
        ");

&Delay ("out          = db_8_byte_0,
        in            = db_8_continue_byte_1 || db_8_state_byte_1,
        enable        = bus_cycle_end,
        reset         = ,
        ");

&PBM_Assign ("db_8_byte_3", "db_8_full_start      ||
                           db_8_state_byte_3      ");
&PBM_Assign ("db_8_byte_1", "db_8_half_start      ||
                           db_8_state_byte_1      ||
                           db_8_continue_byte_1 ");

&PBM_Wire ("db_8_wait_asserted", 1,
          "db_8_byte_3 || db_8_byte_2 || db_8_byte_1");

```



```

# make available to shared busses which contain -both- dynamically-sized
# 8- and 16-bit peripherals (I expect most systems will have no such
# bus, but we compute the address for it here, anyhow).
#
5 # Of course, these addresses only differ in the low bit(s)
# from the original address.
#
my $sys_addr_signal = &Get_Sys_Signal ($$Sys{master}, "address");

10 if (scalar(@dbs_8_modules))
{
    # We alter the low 1 or 2 address bits, depending on whether this
    # is a 16- or 32-bit system:
    #
15 if ($$Sys{master_data_width} == 16)
    {
        $address_lsbs_width = 1;
        &Mux ("dbs_8_address_lsbs, type=unary, declare=1, w=1,
20             dbs_8_byte_1 --> 1'b1,
             dbs_8_byte_0 --> 1'b0,
             --> $sys_addr_signal\[0],
             ");
    } else { # 32-bit system
        $address_lsbs_width = 2;

25         &Mux ("dbs_8_a0, type=unary, declare=1, w=1,
             dbs_8_byte_3 --> 1'b1,
             dbs_8_byte_2 --> 1'b0,
             dbs_8_byte_1 --> 1'b1,
30             dbs_8_byte_0 --> 1'b0,
             --> $sys_addr_signal\[0],
             ");

        # For 32-bit systems, we want to leave bit 1 of the address
        # alone whenever this is a "narrow_16" access.
        &Mux ("dbs_8_a1_raw, type=unary, declare=1, w=1,
35             dbs_8_byte_3 --> 1'b1,
             dbs_8_byte_2 --> 1'b1,
             dbs_8_byte_1 --> 1'b0,
             dbs_8_byte_0 --> 1'b0,
40             --> $sys_addr_signal\[1],
             ");

        &Mux ("dbs_8_a1, type=unary, declare=1, w=1,
45             write_is_narrow_16 --> $sys_addr_signal\[1],
             --> dbs_8_a1_raw");

        &PBM_Wire ("dbs_8_address_lsbs", 2, "{dbs_8_a1, dbs_8_a0}");
    }
50
    my $high_range = "$$Sys{master_address_width} - 1 : $address_lsbs_width";
    my $high_segment= "$sys_addr_signal \[$high_range]";
    &PBM_Assign ("dbs_8_address", "{ $high_segment, dbs_8_address_lsbs}");
55
}

if (scalar(@dbs_16_modules))
{
    # This must be a 32-bit system, and we alter address-bit 1.
    # Note that, by definition, this is a halfword-aligned
60 # address, so A[0] is niether computed nor distributed
    # with this address.
    #
    &Mux ("dbs_16_address_lsb, type=unary, w=1, declare=1,

```

```

        dbs_16_halfword_1 --> 1'b1,
        dbs_16_halfword_0 --> 1'b0,
                                --> $sys_addr_signal\[1],
    ");
5
    &PBM_Assign ("dbs_16_address", "{
        $sys_addr_signal \[$$Sys{master_address_width}-1 : 2],
        dbs_16_address_lsb}");
10
}

# Create a verison of "adjusted" address suitable for any occasion,
# no matter what dynamic operation is (or is not) in process. This
# address value gets distributed on shared address ports.
#
15
# Note that, at least, this mux does collapse to something simpler
# (trivial) if we don't have any dynamic peripherals, or if one class
# (8/16) is entirely missing. That's thanks to the "1'b0" trick, above.
#
20
&Mux ("dbs_adjusted_address,
    type = unary,
    w     = $$Sys{master_address_width},
    dbs_8_active -->    dbs_8_address,
    dbs_16_active --> ({dbs_16_address, 1'b0}),
                                --> $sys_addr_signal,
25
    ");

#####
# Read-data -- registers and assembled-output.
#
30
&PBM_Wire ("bus_cycle_data_ready"); # declared here, computed later.

if (scalar(@dbs_16_modules))
{
    35
    &Delay ("out      = held_halfword_1,
        in      = dbs_16_muxed_input,
        w      = 16,
        enable = (dbs_16_halfword_1 && bus_cycle_data_ready),
        reset  = ,
        ");
    40
    &PBM_Assign ("dbs_16_output", "{held_halfword_1, dbs_16_muxed_input}");
}

if (scalar(@dbs_8_modules))
{
    45
    &Delay ("out      = held_byte_3,
        in      = dbs_8_muxed_input,
        w      = 8,
        enable = (dbs_8_byte_3 && bus_cycle_data_ready),
        reset  = ,
    50
        ")
        if ($$Sys{master_data_width} == 32);

    &Delay ("out      = held_byte_2,
        in      = dbs_8_muxed_input,
    55
        w      = 8,
        enable = (dbs_8_byte_2 && bus_cycle_data_ready),
        reset  = ,
        ")
        if ($$Sys{master_data_width} == 32);
    60

    &Delay ("out      = held_byte_1,
        in      = dbs_8_muxed_input,
        w      = 8,

```

```

        enable = (dbs_8_byte_1 && bus_cycle_data_ready),
        reset = ,
    );

```

```

5      if ($$Sys{master_data_width} == 32)
    {
        &PBM_Assign ("dbs_8_output",
            "{held_byte_3, held_byte_2, held_byte_1, dbs_8_muxed_input}");
    } else {
10        &PBM_Assign ("dbs_8_output",
            "{
                held_byte_1, dbs_8_muxed_input}");
    }
} # End: if (scalar(@dbs_8_modules))

```

```

15 #####
# Write-data
#
# Selection-mux to pick the correct segment to send.
# This is all pretty easy, once you've got the write-control
20 # signals (dbs_8_byte_3, etc) all figured-out.
#
# For now, the read- and write- phase control signals (e.g.
# dbs_8_byte_3 and friends) are the same. In the future, it might
# be beneficial to remove the "narrow-write" exclusion-term from the
# read-phase-control signals, but leave it in the
25 # write-phase-control signals. For today, narrow writing is a
# term in both phase-control signals--BECAUSE THEY'RE THE SAME:
#
my $sys_write_data = &Get_Sys_Signal ($$Sys{master}, "writedata");
30
if (scalar(@dbs_16_modules))
{
    &PBM_Wire ("dbs_16_write_halfword_1", 1, "dbs_16_halfword_1");

    &Mux ("dbs_16_write_data, type = unary, w = 16,
        dbs_16_write_halfword_1 --> ($sys_write_data\[31:16]),
        --> ($sys_write_data\[15:0]),
        ");
}
40
if (scalar(@dbs_8_modules))
{
    if (($$Sys{master_data_width} == 32)) {
        &PBM_Wire ("dbs_8_write_byte_3", 1, "dbs_8_byte_3");
        &PBM_Wire ("dbs_8_write_byte_2", 1, "dbs_8_byte_2");
45    }
    &PBM_Wire ("dbs_8_write_byte_1", 1, "dbs_8_byte_1");

    my $mux_string = "";
    $mux_string .= "dbs_8_write_byte_3 --> ($sys_write_data\[31:24]),
        dbs_8_write_byte_2 --> ($sys_write_data\[23:16]), "
        if $$Sys{master_data_width} == 32;
    $mux_string .= "dbs_8_write_byte_1 --> ($sys_write_data\[15: 8]),
        --> ($sys_write_data\[ 7: 0]), ";
55
    &Mux ("dbs_8_write_data, type = unary, w = 8, $mux_string");
}

# Create a verison of "adjusted" write-data suitable for any occasion,
# no matter what dynamic operation is (or is not) in process. This
# write-data value gets distributed on shared data busses.
#
# Note that, at least, this mux does collapse to something simpler
60

```

09880106-061201

```

# (trivial) if we don't have any dynamic peripherals, or if one class
# (8/16) is entirely missing. That's thanks to the "1'b0" trick, above.
#

```

```

&Mux ("dbs_adjusted_write_data,
5      type = unary,
      w      = $$Sys{master_data_width},
      dbs_8_active --> dbs_8_write_data,
      dbs_16_active --> dbs_16_write_data,
      --> $sys_write_data,
10      ");

```

```

#####
# PBM_Emit_IRQ_Prioritizer
15 #

```

```

# Well, compared to the dynamic bus-sizer, this is sure a piece
# of cake.
#

```

```

20 # Given a %Sys-hash (reference) , we have to emit the requisite
# IRQ prioritization logic into the currently-open file (which, we
# presume, is the PBM verilog file).
#

```

```

# We are responsible for driving two signals:
#

```

```

25 #      1) The masters' "irq"-type input.
#      2) The msters' "irqnumber"-type input.
#

```

```

# (1) is just a straight logical-or of all the periphs' IRQ-outputs.
# (2) is just an ordered priority-mux.
30 #

```

```

#####
sub PBM_Emit_IRQ_Prioritizer

```

```

(
35   my ($Sys) = (@_);
   &Emit_Comment ("\n   IRQ Prioritizer  \n");

```

```

   my @irq_signals = ("1'b0");   # Trick makes it work when list is empty.
   my %irq_hash;   # Keeps track of irq signals by number, so we can sort.

```

```

40   foreach $Mod (&Get_Sys_Slave_List($Sys))
   {
       next if !$Mod{Has_IRQ};           # that was a short trip.

```

```

45       my $irq_signal           = &Get_Sys_Signal($Mod, "irq");
       $irq_hash{$Mod{IRQ_Number}} = $irq_signal;
       push (@irq_signals,
           $irq_signal);
   }

```

```

   &PBM_Assign (&Get_Sys_Signal ($$Sys{master}, "irq") ,
50       join (" || ", @irq_signals) );

```

```

# NOTE: We do not use the generator-library &Mux-function
# because it doesn't retain the order of the terms. That's just
# the way it works. That's OK: It's easy enough to just build
55 # our own question-mark-colon expression:
#

```

```

my $mux_expression_string = "";
foreach $irq_num (sort numerically keys(%irq_hash))
60 {
    $mux_expression_string .= "$irq_hash{$irq_num} ? 6'd$irq_num : ";
}

```

```

$mux_expression_string .= " 6'd63";   # Default: lowest priority.

```

09380105 "061201

```

&PBM_Assign (&Get_Sys_Signal ($$Sys(master), "irqnumber"),
             $mux_expression_string);

5      # Mien Got, that was easy.
    }

#####
# PBM_Emit_Simple_Assignments
10  #
# At the top of the PBM module, there are a bunch of "simple"
# assignments. These are, for the most part, statements which
# "broadcast" master-outputs to various slave modules. For example,
# the master control signals are "simply" assigned to many of the
15  # slave modules.
#
# This function handles this issue role-by-role.
#
# Note that no special care is needed for shared busses, because
20  # we've defined the utility function &PBM_Assign so that you can
# call it multiple times to assign the same thing to the same
# signal, and it does something smart (ignores redundant assignments).
# Thus, a shared byteenable signal, for example, will get the masters'
# byteenable-output assigned to it multiple times, but it won't hurt
25  # anything.
#
# We -don't- connect writedata- or address-type ports for dynamically-sized
# peripherals. Those are generated in the DBS-unit. For the same reason,
# we also don't assign address- or writedata- signals to shared busses,
30  # because one of the clients might require dynamic sizing.
#
#####
sub PBM_Emit_Simple_Assignments
(
35      my ($Sys) = (@_);

# Lots of things connect to the master, so let's keep a ref to the
# master's data, and its avalon-ports, handy:
my $Master_Mod = $$Sys(master);
40  my $master_avalon_table = $$Master_Mod{avalon_port_table};

&Emit_Comment("\nSimple assignments.
Connect-up signals which pass unmolested from one PBM-port to another.");

45  &PBM_Wire ("reset", 1, "~reset_n"); # Handy: Logic-true reset.

#####
# Start with the very-easiest avalon roles:
#     always0, always1, clk, and resetn
50  #
# Even these are a little abnormal, because we have to deal with the
# fact that there could be more than one of each type.
#
foreach $Mod (&Get_Sys_Module_List($Sys)) {
55      foreach $Port (&Get_Module_Port_List ($Mod))
      {
          if ($$Port{avalon_role} eq "always0") {
              &PBM_Assign ($$Port{system_signal}, "({$Port{width}}'b0)");
          } elsif ($$Port{avalon_role} eq "always1") {
              &PBM_Assign ($$Port{system_signal}, "({$Port{width}}'b1)");
60          } elsif ($$Port{avalon_role} eq "clk") {
              &PBM_Assign ($$Port{system_signal}, "clk");
          } elsif ($$Port{avalon_role} eq "resetn") {

```

```
&PBM_Assign ($$Port{system_signal}, "reset_n");
```

```
}
```

```
}
```

```
}
```

```
#####
```

```
# Address-broadcast.
```

```
#
```

```
# Each peripheral with an address-port gets the appropriate  
# flavor of address -- except, of course, dynamically-sized peripherals,  
# which get a special address generated by the DBS-unit.
```

```
#
```

```
# The byte-enable control signals could, I suppose, be considered "part  
# of" the address, so they get assigned in this same loop. Once  
# again, byte-enables for dynamic devices are handled elsewhere.
```

```
#
```

```
# !!!SUBOPTIMALITY NOTE!!!
```

```
#
```

```
# Shared-busses are complicated, because it's a bit of an ordeal to  
# figure-out exactly -which- flavor of dynamic address they get--it  
# depends upon what's connected to them. If we were to do this in  
# some clever, optimal way, it would be a big hassle (trust me: a  
# great big hassle).
```

```
#
```

```
# Instead, we always assign a dbs-adjusted  
# version of the address (which comes from logic) to shared address  
# ports. Even shared address ports that don't have any dynamically-sized  
# clients on them. Thus, we sometimes introduce more logic-delay than  
# strictly necessary. Sue me. I'll fix it if it ever becomes a  
# problem.
```

```
#
```

```
my $byte_range_select = "$$Sys{master_address_width}-1 : 0";  
my $halfword_range_select = "$$Sys{master_address_width}-1 : 1";  
my $word_range_select = "$$Sys{master_address_width}-1 : 2";  
&Emit_Comment ("\n Address-assignments\n");
```

```
# Emit wire-declarations for DBS-generated addresses, whether we  
# need them or not. These get assigned-to when we build the  
# dynamic bus-sizers, but we (might) use them here:
```

```
#
```

```
&PBM_Wire ("dbs_16_address", $$Sys{master_address_width}-1);  
&PBM_Wire ("dbs_8_address", $$Sys{master_address_width} );  
&PBM_Wire ("dbs_adjusted_address", $$Sys{master_address_width} );
```

```
#
```

```
foreach $Mod (&Get_Sys_Slave_List($$Sys))
```

```
{
```

```
    next if $$Mod{Is_Bus_Master}; # Slaves only, please
```

```
    #
```

```
    # Go ahead and hook-up the byte-enables. That's straightforward.
```

```
    #
```

```
    # Note: I may need to change this later to work with narrow writes  
    # through the DBS-unit.
```

```
    #
```

```
&PBM_Assign (&Get_Sys_Signal ($Mod, "byteenablen"),  
             &Get_Sys_Signal ($Master_Mod, "byteenablen"));
```

```
    #
```

```
    my $source_signal = "";  
    my $A_Port = &Get_Port_By_Role ($Mod, "address");  
    next if !$A_Port;
```

```
    #
```

```
    if ($$A_Port{is_shared})
```

```
    {
```

09280105-061201

```

# All shared addresses are dbs-muxed.  How suboptimal.
$source_signal = "dbs_adjusted_address";
}
elseif ($$Mod{is_dynamically_sized})
5 {
    $source_signal =
        $$Mod{address_type_used} eq "byte" ? "dbs_8_address" :
        "dbs_16_address" ;
} else { # Non-dynamic address
10
    my $range_select =
        $$Mod{address_type_used} eq "byte" ? $byte_range_select :
        $$Mod{address_type_used} eq "halfword" ? $halfword_range_select :
        $word_range_select ;
15
    $source_signal =
        &Get_Sys_Signal ($Master_Mod, "address") . "[$range_select]";
}

&PBM_Assign (&Get_Sys_Signal ($Mod, "address"), $source_signal);
20 }

#####
# Data-broadcast
#
25 # Pretty simple: Each peripheral gets a copy of the master's write-data,
# unless it's dynamic--then it gets a special "tweaked" version of the
# data.
#
# Shared data-ports could, in theory, have any kind of peripheral attached
30 # to them, so they need to drive-out a fully "dynamically-adjusted"
# version of the data suitable for the current occasion. Happily, the
# DBS-unit computes this very thing for us.
#
# See the !!!SUBOPTIMALITY NOTE!!! above for addresses.
35 # The same thing goes here. We're penalizing the write-data path on
# busses that may not have any dynamic clients at all. For now:
# tough beans. For later, maybe we'll get more clever.
#
# Another slight suboptimality: The way this works, we drive data out
40 # on -all- tri-state busses whenever the master does a write. This
# might involve a slight waste of power, because we'll be wiggling I/O
# pins more than strictly necessary. Again, tough beans.
#
&Emit_Comment ("\n Data-assignments\n");
45

# Emit wire-declarations for DBS-generated write-data, whether we
# need them or not. These get assigned-to when we build the
# dynamic bus-sizers, but we (might) use them here:
#
50 &PBM_Wire ("dbs_16_write_data", 16);
&PBM_Wire ("dbs_8_write_data", 8);
&PBM_Wire ("dbs_adjusted_write_data", $$Sys{master_data_width});

foreach $Mod (&Get_Sys_Slave_List($Sys))
55 {
    my $source_signal = "";
    my $target_signal = &Get_Sys_Signal ($Mod, "writedata");

    if ($$Mod{Uses_Tri_State_Data_Bus})
60 {
        # The "principal" tri-state bus is managed as part of the
        # read-data path (elsewhere).
        next if $$Mod {Tri_State_Data_Bus} eq

```

\$\$\$Sys{Principal_Tri_State_Data_Bus} ;

For tri-state busses, we need to use the "data"-type port
instead of the "writedata"-type port as the assignment
target:

\$target_signal = &Get_Sys_Signal (\$Mod, "data");

To this bus we will assign a value which, as it
happens, incorporates its whole tri-state logic. Perversely,
we may "PBM_Assign" this same expression to this bus several
times, but it doesn't hurt anything.

my \$sys_writen = &Get_Sys_Signal (\$\$Sys{master}, "writen");
\$source_signal = "(~\$sys_writen) ? dbs_adjusted_write_data :
 \$\$\$Sys{master_data_width}'bZ ";

}
elseif (\$\$Mod{is_dynamically_sized})
{

 \$source_signal =
 \$\$Mod{address_type_used}.eq "byte" ? "dbs_8_write_data" :
 "dbs_16_write_data" ;

} else { # Non-dynamic address
 \$source_signal = &Get_Sys_Signal (\$Master_Mod, "writedata");
}

&PBM_Assign (\$target_signal, \$source_signal);

#####

Readn/Writen -broadcast.

Every module gets a copy of the master's "readn" and "writen" signals
-- unless they have a nonzero setup- or hold-time, in which case
they get their very-own custom-made, tweaked copy of these signals.

That all happens later when the wait-state unit gets created.

&Emit_Comment ("\\n Control-assignments\\n");

foreach \$Mod (&Get_Sys_Slave_List(\$Sys))

{
 next if \$\$Mod{Setup_Time} != 0;
 next if \$\$Mod{hold_time_full_clocks} != 0;

 &PBM_Assign (&Get_Sys_Signal (\$Mod, "ifetch"),
 &Get_Sys_Signal (\$Master_Mod, "ifetch"));

 &PBM_Assign (&Get_Sys_Signal (\$Mod, "readn"),
 &Get_Sys_Signal (\$Master_Mod, "readn"));

 &PBM_Assign (&Get_Sys_Signal (\$Mod, "writen"),
 &Get_Sys_Signal (\$Master_Mod, "writen"))
 unless \$\$Mod{Hold_Time}; # half-cycle hold needs custom write strobe.

#####

PBM_Emit_Address_Decoder

#

Given a ref to the %Sys-hash, we can generate all the logic
to build the address decoder.

#

This function emits all the Verilog statements which

09330106 "061201


```

# implement the address-decoder directly into the currently-selected
# output file. These statements include:
#
# * wire-declarations for modules that don't have chip-select ports.
5 # * Logic-assignments to all "active-" signals.
# * Instantiate fast I/O registers for "registeredselectn"s, if any.
#
#####

10 sub PBM_Emit_Address_Decoder
{
    my ($Sys) = (@_);

    &Emit_Comment ("\n    Address decoder \n");

15 # Extract the name of the masters' address signal:
    my $Master_Mod = $$Sys{master};
    my $master_address = &Get_Sys_Signal($Master_Mod, "address");

20 # external chip selects may get gated with slave_phase below.
    &Vprint("reg slave_phase;\n");

    foreach $Mod (&Get_Sys_Slave_List($Sys))
    {
25        $$Mod{external_active_signal} = &Get_Sys_Signal ($Mod, "chipselect");

        # All modules get a pbm_internal select signal regardless of if they
        # need an external chip select signal. This is because
        # we use address-decode signals to operate the
30        # read-data mux and wait state timer. PBM_external signals have
        # gotten a tad more tricky. We now gate them with slave_phase.
        # This may
        #
        # All modules, including ones requiring the evil
        # "registeredselectn"-type signals also fall into this category.

        $$Mod{internal_active_signal} = "$$Mod{name}_active";

        # How quaint that we still call this old warhorse of a function:
40        #
        &Make_Nios_Chip_Select ("out
                                device_base = $$Mod{Base_Address},
                                device_span = $$Mod{address_span},
                                outer_span  = $$Sys{address_span},
                                address_name = $master_address,
45                                ");

        if ($$Mod{external_active_signal})
        {
50            if (
                ($$Mod{Read_Wait_States} eq "peripheral_controlled") ||
                ($$Mod{Write_Wait_States} eq "peripheral_controlled")
            )
            {
55                &PBM_Wire ($$Mod{external_active_signal},1,
                            "$$Mod{internal_active_signal} & slave_phase");
            }
            else
            {
60                &PBM_Wire ($$Mod{external_active_signal},1,
                            $$Mod{internal_active_signal});
            }
        }
    }
}

```

00000106:05:1201

}

#####

Registered chip-selects.

#

The whole dirty little business of registered chip-select signals
is, I suppose, part of the address-decoding job. Note
that modules can have more than one registered chip-select signal,
each of which is derived (of course) from that modules' active-signal.

#

There's also a wait-state aspect to this task (wait-states
occur as a consequence of the chip-select delays). That's not
handled here--it's handled later, when we build the wait-state
unit.

#

my \$Fast_IO_Setting = "OFF";

my \$chipselect_reg_module = \$\$Sys{name} . "_rg";

if (\$\$Sys{has_registered_select_signals})

{

\$Fast_IO_Setting = "ON";

First, we create a special flip-flop module just for the
purpose. It needs to be "firm" so that synthesis tools don't
optimize-away duplicates, and so that we can make a
FAST_OUTPUT_REGISTER entity-assignment to it without "losing" its
name.

#

&Create_Firm_Flip_Flop_Variant (\$\$Sys{system_directory},
\$\$Sys{sopc_directory},
\$\$Sys{device_family},
\$chipselect_reg_module,
);

Push this register onto the list of files to be synthesized.

my \$synth_list_ref = \$\$Sys{synth_file_list};

push (@\$synth_list_ref,
"\$Sys{system_directory}/\$chipselect_reg_module.v");

#&Emit_Comment

(q[

Registered chip-select signals

A module may optionally request one (or more) -registered- chip-select
signals (by declaring a port of type "registeredselectn", and by
setting the "Uses_Registered_Select_Signal" assignment TRUE in the
SYSTEM_BUILDER_INFO section).

Modules do this when they have stringent setup-time requirements
on their select-signals. Such stringent setup requirements are
greatly assisted by having the select-signal be produced from
an Apex Fast-I/O register, right in the pad structure itself.

Delaying (registering) the select-signal to a module is tricky
business, but the PBM "absorbs" all the trickiness and does the right
thing. Wait-states get generated (in subsequent code) when the
(delayed) select-signal being fed to the device disagrees with its
correct, current value. The logic is clever enough to allow successive
accesses to this same device with no wait-penalties. This
arrangement is eminently suitable for external asynchronous RAM
used as main-memory.

Registered chip-selects are always logic-negative, because that's the

00000106 061201
T02190" 90T08820

way chip-level select signals have been since the dawn of time. Because these come directly from fast I/O registers, there is no subsequent opportunity to negate them, or even copy them to other pins.

We accomplish all this using deep voodoo magic: We use a special "Firm Flip-Flop" module, since this is the only way to convince the synthesis tool to -not- optimize-out "redundant" chip-select registers. It also provides a handy method for assigning the 'FAST_OUTPUT_REGISTER' attribute--using an ESF-file.

```
});
```

```
foreach $Mod (&Get_Sys_Slave_List($Sys)) {
  foreach $Port (&Get_Module_Port_List ($Mod))
  {
```

```
    next if $$Port{avalon_role} ne "registeredselectn";
```

```
    # If we got to here, then we know $$Port is
    # of type "registeredselectn". Blurt-out an instance
    # of a properly-connected firm flip-flop with no further ado:
    #
```

```
    my $instance_name = "$$Mod{name}_$$Port{name}_delay_register";
    &Vprint ("
```

```
        $chipselect_reg_module $instance_name
```

```
    (
```

```
        .clk      (clk),
```

```
        .ena      (1'b1),
```

```
        .clrn     (reset_n),
```

```
        .prn      (1'b1),
```

```
        .d        (~$$Mod{internal_active_signal}),
```

```
        .q        ($$Port{system_signal})
```

```
    );
```

```
    // exemplar attribute $instance_name NOOT TRUE
```

```
    ");
```

```
  }
```

```
}
```

```
} # End: if ($$Sys{has_registered_select_signals})
```

```
# We -always- create an ESF file because:
```

```
# 1) don't cost nuthin'
```

```
# 2) Un-sets FAST-I/O req'mnt if it had been previously set.
```

```
#
```

```
&Create_ESF_File
```

```
    ("firm_flip_flop : FAST_OUTPUT_REGISTER = $Fast_IO_Setting;",
```

```
        $$Sys{system_directory},
```

```
        $chipselect_reg_module);
```

```
}
```

```
#####
```

```
# PBM_Emit_Read_Data_Mux
```

```
#
```

```
# Given a ref to the %Sys-hash, we can generate all the logic
```

```
# to build the read-data mux path.
```

```
#
```

```
# This function emits all the Verilog statements which
```

```
# implement the read-data mux structure directly into the
```

```
# currently-selected output file.
```

```
#
```

```
# **** Mux structure
```

```
#
```

```
# We build-up a "fast mux" and a "slow mux." The "fast mux"
```

```
# gathers-up all the readdata-type signals for zero-wait-state
```

```

# modules (if any). The slow-mux gathers-up all the others.
#
# **** The "principal" bus.
#
5 # The road to the CPU takes another twist if you have a "principal"
# tri-state data bus. The principal data bus gets routed -directly-
# to the CPU's read-data port--the rest of the (muxed-together) signals
# a route "around the horn": Driven-out onto the principal
# databus in order to be read back in.
10 #
# If the CPU doesn't have any such "principal" tri-state busses, then
# things are simpler--the muxes just collect the data.
#
# **** Widths
15 #
# In this age of dynamic bus-sizing, all narrow data busses are just
# zero-padded. This happens "almost invisibly" in Verilog syntax when
# we assign a narrow value to a wide target.
#
20 # If a peripheral is "dynamic," then we use the ouptut
# from the appropriate DBS-unit in place of its data. The DBS-unit
# itself is generated later. And, to help-out the DBS-unit, we
# also generate its input-muxes right here, because we are already
# engaged in the process of building mux-like things.
25 #
#####
sub PBM_Emit_Read_Data_Mux
{
30   my ($Sys) = (@_);
   &Emit_Comment ("\n   Read-Data Multiplexer \n");

   #####
   # Loop to build-up slow- and fast-mux strings.
   #
35   # Also, build-up mux strings for the dynamic bus-sizers' inputs.
   #
   my $fast_mux_string = "";
   my $slow_mux_string = "";
   my $dbs_8_mux_string = "";
40   my $dbs_16_mux_string = "";

   foreach $Mod (&Get_Sys_Slave_List($Sys))
   {
45     my $data_signal = &Get_Sys_Signal ($Mod, "readdata");
     $data_signal = &Get_Sys_Signal ($Mod, "data")
       if $$Mod{Uses_Tri_State_Data_Bus};

     next if $data_signal eq ""; # Some peripherals are write-only (!)

50     if ($$Mod{is_dynamically_sized})
     {
       # Yikes. A dynamically-sized peripheral. Its data comes from
       # the appropriately-sized DBS-unit. Note that dynamically-sized
       # peripherals are automatically assumed to be "slow."
55       #
       if ($$Mod{Data_Width} <= 8)
       {
         $slow_mux_string      .= "$$Mod{internal_active_signal}  -->
dbs_8_output,";
60         $dbs_8_mux_string    .= "$$Mod{internal_active_signal}  -->
$data_signal,";
       } else {

```

```

5      $slow_mux_string      .=      "$$Mod{internal_active_signal}  -->
dbs_16_output,";
      $dbs_16_mux_string    .=      "$$Mod{internal_active_signal}  -->
      $data_signal,";
    }

    } else {
      # Not dynamic--it's ether a slow-thing or a fast-thing.
      #
10     # Principal data-bus is not (directly) part of this mux.
      # but we are accutely interested in the active-signals
      # for peripherals which sit on the principal bus. We'll
      # use these signals later to control the bus-direction.
      next if $$Mod{Uses_Tri_State_Data_Bus} &&
15      ($$Mod{Tri_State_Data_Bus} eq $$Sys{Principal_Tri_State_Data_Bus});

      if ($$Mod{Read_Wait_States} == 0) {
        $fast_mux_string    .=      "$$Mod{internal_active_signal}  -->
        $data_signal,";
20      } else {
        $slow_mux_string    .=      "$$Mod{internal_active_signal}  -->
        $data_signal,";
      }
    }
  } # End: $Mod-loop.

  # There may or may not be input-muxes to the DBS-units:
  # if so, we declare the DBS-unit's output wire here
  # (becuase we have to). But the DBS-units themselves are made later.
30  #
  if ($dbs_8_mux_string)
  {
    &Modified_Mux("dbs_8_muxed_input, type=unary, w=8, $dbs_8_mux_string");
    &PBM_Wire ("dbs_8_output", $$Sys{master_data_width});
35  }

  if ($dbs_16_mux_string)
  {
    &Modified_Mux("dbs_16_muxed_input, type=unary, w=16,
                  $dbs_16_mux_string");
    &PBM_Wire ("dbs_16_output", $$Sys{master_data_width});
40  }

  # There may or may not be a slow mux:
  #
45  #
  if ($slow_mux_string)
  {
    &Modified_Mux ("slow_read_data_mux_output, type=unary,
                  w = $$Sys{master_data_width},          $slow_mux_string
50      ");

    # The "default" input to the fast-mux is, of course, the output
    # of the slow-mux:
    #
55    $fast_mux_string .= " --> slow_read_data_mux_output,";
  }

  # There is always a fast-mux:
  &Modified_Mux ("read_data_mux_output, type=unary,
60      w = $$Sys{master_data_width},
      $fast_mux_string
      ");

```

```

#####
# Data-in to CPU.
#
# Well, gee. There are two cases here. Let's start with the
5 # easiest:
#
# **** -NO- principal tri-state bus.
#
# In this case, the fast-mux output goes right to the
10 # CPU's data-input, and that's that.
#
# **** Principal tri-state bus.
#
# The principal tri-state bus actually drives the CPU's
15 # read-data input directly. The fast-mux output gets registered.
# The output of this register gets driven-out onto the principal
# bus (and, therefore, to the CPU). Thus, perversely, we -drive-
# data out onto this bus at the end of any bus read-transaction
# which did not get data from the principal bus (and, even then,
20 # we do it if there's dynamic bus-sizing).
#
&PBM_Wire ("data_driveback_active"); # These Declare here, compute later.
&PBM_Wire ("data_driveback_asserted");
&PBM_Wire ("dbs_8_active");
25 &PBM_Wire ("dbs_16_active");

if (!$$Sys{Principal_Tri_State_Data_Bus})
{
    # The easy case: Mux feeds CPU, period.
    #
    &PBM_Assign (&Get_Sys_Signal ($$Sys{master}, "readdata"),
30                 "read_data_mux_output");
} else {
    # Ugh.
35
    # To start with, hook-up principal bus directly to CPU's data-in:
    #
    &PBM_Assign (&Get_Sys_Signal ($$Sys{master}, "readdata"),
40                 "$$Sys{Principal_Tri_State_Data_Bus}_data");

    # Next, run the result of the read-data mux into a holding
    # register:
    &Delay("in          = read_data_mux_output,
45           w          = $$Sys{master_data_width},
           reset       = ,
           ");

#####
# What to drive?
50 #
# We can drive one of two things out on the principal data
# bus:
# 1) Outbound write-data from the CPU.
#     For this purpose, we use the same "dbs_adjusted_write_data"
55 #     that every other tri-state bus gets. This is suboptimal
#     in the same way that it is for every other tri-state bus.
#
# 3) Data from the read-mux being sent "back out" to the CPU.
#
60 # Those are the only two possibilities, and we can tell which
# to use just by looking at the masters' written-signal:
#
my $master_write_n = &Get_Sys_Signal ($$Sys{master}, "written");

```

```

&Mux ("principal_data_drive_value,    type=unary, declare=1,
      w = $$Sys(master_data_width),
      (~$master_write_n) --> dbs_adjusted_write_data,
      --> dl_read_data_mux_output,
      ");

```

```

#####

```

```

# When to drive?

```

```

#
# Like every other tri-state bus, we drive data during actual
# bona-fide write-operations. We also drive data during the
# "driveback" phase, as-determined by a signal named:

```

```

#
#     data_driveback_asserted

```

```

#
# which (we presume) gets computed by the wait-state generator.

```

```

#
# **** Interaction: Dynamic Bus Sizing

```

```

#
# And, of course, there's a bit of strangeness when a device
# on the principal databus is dynamically-sized. The easiest
# way to think about this: Set aside dynamic-sizing for a
# moment, and think only about narrow peripherals on the
# principal databus (be they dynamically sized or no).

```

```

#
# Narrow peripherals, by definition, only drive low
# bits (e.g. LS-byte or LS-halfword) of the databus. So,
# when reading from (say) an 8-bit device on a tri-state databus,
# the high-order bits [31..8] aren't driven by anyone. There's
# no harm in -us-driving these bits, now, is there? If we
# don't, then they'll just read as <undefined> ('Z', in
# simulation), and that's not really any help to anyone.

```

```

#
# So let's suppose we decide to be good citizens and "nail-down"
# the high-order address bits when a narrow peripheral is
# accessed on the principal tri-state bus. This gives us a warm
# feeling inside, and, as an added bonus, makes dynamic bus
# sizing work for devices on the principal bus (the high-bits
# get driven from the appropriate dbs-holding registers. Yay.

```

```

#
# So, despite the heft of this comment, the actual solution is
# both simple and tidy:

```

```

&PBM_Wire ("do_drive_principal_data_bus_byte_0", 1,
          "~$master_write_n || data_driveback_active");

```

```

&PBM_Wire ("do_drive_principal_data_bus_byte_1", 1,
          "do_drive_principal_data_bus_byte_0 || dbs_8_active");

```

```

&PBM_Wire ("do_drive_principal_data_bus_bytes_2_and_3", 1,
          "do_drive_principal_data_bus_byte_0 ||
          dbs_16_active || dbs_8_active");

```

```

# Assign principal databus in bitwise-chunks.

```

```

#
&PBM_Assign("$$Sys(Principal_Tri_State_Data_Bus)_data[7:0]",
            "(do_drive_principal_data_bus_byte_0    ?
             principal_data_drive_value[7:0]       :
             8'bZ                                   )
            ");

```

09860106 061201

```

&PBM_Assign("$$Sys{Principal_Tri_State_Data_Bus}_data[15:8]",
    "(do_drive_principal_data_bus_byte_1 ?
        principal_data_drive_value[15:8] :
        8'bZ
    ");

```

```

if ($$Sys{master_data_width} == 32) {
    &PBM_Assign("$$Sys{Principal_Tri_State_Data_Bus}_data[31:16]",
        "(do_drive_principal_data_bus_bytes_2_and_3 ?
            principal_data_drive_value[31:16] :
            16'bZ
        ");
}

```

```

#####

```

```

# "memis32bits"

```

```

#

```

```

# Somebody needs to tell the master when it's fetching

```

```

# 32-bit data--that's just one of the inputs that an Avalon master
# might require.

```

```

#

```

```

# I couldn't say for sure that this really belongs in the "Read Data Mux"
# generator, but I can't think of where else to put it, so here
# it is.

```

```

#

```

```

# Note that 16-bit dynamically-sized devices appear as 32 bits
# -unless- "ifetch" is asserted, in which case they don't. 8-bit
# dynamically-sized devices always just appear as 32-bits wide.

```

```

#

```

```

# It might be more efficient to say when something -doesn't- return
# a 32-bit value. But For now, I do the logic-true computation:

```

```

#

```

```

my $ifetch_signal = &Get_Sys_Signal ($$Sys{master}, "ifetch");

```

```

my @memis32bits_terms = ("1'b0");

```

```

foreach $Mod (&Get_Sys_Slave_List ($Sys))

```

```

{

```

```

    if ($$Mod{Data_Width} > 16) {
        push (@memis32bits_terms, $$Mod{internal_active_signal});
    } else {

```

```

        if ($$Mod{is_dynamically_sized}) {
            if ($$Mod{Data_Width} <= 8) {

```

```

                push (@memis32bits_terms, $$Mod{internal_active_signal});
            } else {

```

```

                push (@memis32bits_terms,
                    "$$Mod{internal_active_signal} && (~$ifetch_signal)");
            }

```

```

        }

```

```

    }

```

```

}

```

```

&PBM_Assign (&Get_Sys_Signal ($$Sys{master}, "memis32bits"),
    join (" || ", @memis32bits_terms));

```

```

}

```

```

#####

```

```

# PBM_Emit_Wait_State_Generator

```

```

#

```

```

# Given a ref to the %Sys-hash, we generate all the logic
# to build the read-data mux path. The logic gets
# dumped-into the currently-selected output file.

```


09880105 "051201
T0219866

```
#
#####
sub PBM_Emit_Wait_State_Generator
{
5   my ($Sys) = (@_);
    &Emit_Comment ("\n    Wait-State Generator  \n");

    # This function seems preternaturally-concerned with the read-
    # and write-strobes.  Assign their signal-names to some
10   # handy local variables:
    #
    my $sys_readn = &Get_Sys_Signal ($$Sys{master}, "readn");
    my $sys_writen = &Get_Sys_Signal ($$Sys{master}, "writen");

15   #####
    # Peripheral-Controlled Wait
    #
    # If the currently-selected peripheral has a wait-request
    # signal, and it's asserting it, then we definitely want to
20   # wait.  This is pretty easy to compute:
    #
    my @periph_controlled_wait_terms = ("1'b0"); # 1'b0 fixes empty lists.

25   foreach $Mod (&Get_Sys_Slave_List($Sys))
    {
        my $request_signal = &Get_Sys_Signal ($Mod, "waitrequest");

        #####
        # @periph_controlled_wait_terms is one of the very few things
        # (only one at the moment) that gets an
        # external_active_signal.  We cut off the external select
        # after the peripheral lowers its wait pin.  It doesn't get to
        # change its mind later on in the same bus cycle.

35         push (@periph_controlled_wait_terms,
                "$$Mod{external_active_signal}      &&      $request_signal      &&
(~$sys_writen)"
                ) if $$Mod{Write_Wait_States} eq "peripheral_controlled";

40         push (@periph_controlled_wait_terms,
                "$$Mod{external_active_signal} && $request_signal && (~$sys_readn)"
                ) if $$Mod{Read_Wait_States} eq "peripheral_controlled";

45     }

    &PBM_Wire ("periph_wait_asserted", 1,
               join (" || ", @periph_controlled_wait_terms));

50   #####
    # Calculate minimum wait states.
    #
    # Each module has a certain minimum number read/write wait-states,
    # perhaps even including zero.  This is the sum of the
55   # user-supplied wait-states and, IN ADDITION,
    # any setup- and hold-times the module may require.
    #
    # It may seem a bit strange, but even modules with
    # "peripheral_controlled" wait-states can also have a
60   # minimum number of fixed wait-states--that's because they may
    # also require setup/hold time.
    #

```

```

# The minimum number of wait-states we calculate here (and save
# as part of each %Mod-hash) DOES NOT INCLUDE delays due to
# dynamic bus sizing or principal-databus driveback.
#
5 # Here we just compute the minimum number of clocks -In A Single
# Bus Cycle-. If the peripheral requires multiple bus cycles
# (via dynamic bus sizing) or requires an extra clock for databus
# drive-back, that's not our problem here.
#
10 # The values we compute here (minus 1) get loaded into the
# wait-state counter when any of these peripherals are selected.
#
# while we're looping through the values, take note of the
# largest number we'll see--this number (less 1) will be the biggest
15 # value we ever load into the wait-state counter.
#
# **** Hold-time policy:
#
# Giving a peripheral extra hold-time doesn't make much sense
20 # following a read-cycle--though one can contrive a case where
# you need it. We could define it either way-- hold-times apply
# only to write-cycles, or they apply to both read- and write-cycles.
# I've defined it so that hold-times only apply to write-cycles. It
# makes computation of the "bus_cycle_data_ready" signal simpler.
25 #
# Deal with nasty case of no wait-states whatsoever, which will
# result in a zero-bit mux/counter. To deal with that case, we
# just always assume at least one wait-state:
30 #
my $max_counter_value = 1;

foreach $Mod (&Get_Sys_Slave_List($Sys))
{
35   my $min_read = $$Mod {Read_Wait_States};
   my $min_write = $$Mod {Write_Wait_States};

   $min_read = 0 if $min_read eq "peripheral_controlled";
   $min_write = 0 if $min_write eq "peripheral_controlled";

40   $min_read += $$Mod {Setup_Time};
   $min_write += $$Mod {Setup_Time} + $$Mod {hold_time_full_clocks};

   $$Mod {read_min_wait_states} = $min_read;
   $$Mod {write_min_wait_states} = $min_write;
45   $max_counter_value =
       &Find_Max ($max_counter_value, $min_read, $min_write);
}

50 my $wait_counter_bits = &Bits_To_Encode($max_counter_value);

#####
# Counter-load mux
55 #
# Select what value goes into the wait-state counter
# based on the peripherals' active-signals (and, of course,
# whether we're reading or writing).
#
60 # Notice that we load the counter with the modules' number of
# wait-states MINUS ONE.
#
# Trick: All these muxes are very similar, so we use the same

```

090901060601201

```

#      description-string with a different word at the front,
#      which gives each a different output-signal:
#

```

```

5 my $counter_mux_string = "counter_load_value,
                           w      = $wait_counter_bits,
                           type   = unary,
                           --> 0,
                           ";

```

```

10 my $write_counter_mux_string = "write_" . $counter_mux_string;
    my $read_counter_mux_string = "read_" . $counter_mux_string;

```

```

    foreach $Mod (&Get_Sys_Slave_List($Sys))
    {

```

```

15     my $read_load_val = $$Mod {read_min_wait_states} - 1;
        my $write_load_val = $$Mod {write_min_wait_states} - 1;

```

```

        $read_counter_mux_string      .=      "$$Mod {internal_active_signal}  -->
$read_load_val,"
20         if ($read_load_val >= 0);
        $write_counter_mux_string     .=      "$$Mod {internal_active_signal}  -->
$write_load_val,"
        if ($write_load_val >= 0);
    }

```

```

25     &Modified_Mux ( $read_counter_mux_string);
        &Modified_Mux ($write_counter_mux_string);

```

```

30     &Mux (" $counter_mux_string, declare=1,
           (~$sys_writen) --> write_counter_load_value,
           (~$sys_readn)  --> read_counter_load_value,
           ");

```

```

35     #####

```

```

    # Wait-state counter.

```

```

    #

```

```

    # There are two interesting questions about this counter:

```

```

    #

```

```

40    # -- When should it load?

```

```

    # -- When should it count?

```

```

    #

```

```

    # **** When to load:

```

```

    #

```

```

45    # This counter gets loaded at the beginning of every bus-cycle
    # for which a fixed wait is required. We already have a signal
    # which tells us when it's the beginning of a bus-cycle, so we
    # need some additional logic to tell us if a fixed-wait is required.
    # We build that logic herein below, before the counter proper.

```

```

    #

```

```

50    # **** When to count:

```

```

    #

```

```

    # This counter "sticks" at zero. This makes the whole scheme
    # much easier to manage/understand. Consequently, the counter
55    # is only enabled when its current value is nonzero.

```

```

    #

```

```

    # The only other thing that can stop this counter from a-countin'
    # is a wait-request from the active peripheral. This
    # has the effect of extending the current Bus Cycle-- setup-time,
60    # hold-time, and all.

```

```

    #

```

```

    # Note that there is a question of interpretation here: Do
    # we want to allow a peripheral to extend a bus-cycle during its

```

09880106.061201

```

# own setup-time? Or should we not listen to it until we actually
# issue it an active read/write strobe? This is largely a question
# of definition. I chose to -always- listen to peripheral-controlled
# waits, even during setup/hold periods. Because it makes the logic
# easier. So there.
#

my @fixed_wait_active_terms = ("1'b0"); # 1'b0 fixes empty-strings.

foreach $Mod (&Get_Sys_Slave_List($Sys))
{
    if ($$Mod{read_min_wait_states} > 0) {
        push (@fixed_wait_active_terms,
            "((~$sys_readn) && $$Mod{internal_active_signal})");
    }
    if ($$Mod{write_min_wait_states} > 0) {
        push (@fixed_wait_active_terms,
            "((~$sys_writen) && $$Mod{internal_active_signal})");
    }
}

# This signal stays true during the entire bus-cycle
# for which a fixed (counter) wait-state applies.
#
&PBM_Wire ("fixed_wait_active", 1, join (" || ", @fixed_wait_active_terms));

# The counter itself, per-se.
#
# I use one call to "&Delay" to build a whole counter. Nobody else
# seems to appreciate the mighty "&Delay" function--the fools.
#
&PBM_Wire ("counter_ne_zero");
&PBM_Wire ("wait_count_enable", 1, "(!periph_wait_asserted) &&
    (counter_ne_zero)");

&PBM_Wire ("bus_cycle_start"); # declared here, computed later.
&PBM_Wire ("wait_load_enable", 1, "bus_cycle_start &&
    fixed_wait_active");

# Optimization: Avoid emitting the counter-register if the maximum
# counter load-value happens to be zero. This often happens
# when we have a bunch of one-wait-state peripherals in the
# system. In this case, the single wait-state is controlled
# by the "wait_load_enable" signal, and we can do away with the
# counter per-se in its entirety. A clever synthesis tool might
# figure this out on its own, but...why risk it?
if ($max_counter_value <= 1) {
    &PBM_Wire ("wait_counter", 1, "1'b0");
} else {
    &Delay ("out          = wait_counter,
        in            = (wait_counter - 1),
        enable        = (wait_load_enable || wait_count_enable),
        sync_set      = wait_load_enable,
        set_value     = counter_load_value,
        width         = $wait_counter_bits,
        reset         = ,
        ");
}
&PBM_Assign ("counter_ne_zero", "wait_counter != $wait_counter_bits'b0");

&PBM_Wire ("fixed_wait_asserted", 1, "counter_ne_zero || wait_load_enable");

#####

```

09830106 "061201

0980106-061201

```
# Registered chip-selects.
#
# This is a nuisance, but at least it's easy.
#
5 # While we're listing all the reasons to extend a bus cycle,
# let us not forget registered chip-selects. They blow!
#
# Whenever the registered (delayed) version of a chip-select
# differs from the current (un-delayed) version, we view
10 # this as a bad thing. We extend the current bus cycle until
# the signals agree (i.e. we wait 1 clock).
#
my @reg_select_wait_terms = ("1'b0"); # Ah, the old "1'b0" trick.

15 foreach $Mod (&Get_Sys_Slave_List($Sys))
{
    next unless $$Mod{Uses_Registered_Select_Signal};

    # Modules -might- have more than one registered select signal,
    # but we don't really care. They all have the same time-behavior.
20 #
    my $reg_signal = &Get_Sys_Signal ($Mod, "registeredselectn");
    push (@reg_select_wait_terms, "(!~$reg_signal)");
    $$Mod{internal_active_signal}++;
25 }

    &PBM_Wire ("reg_select_wait_asserted", 1;
        join (" || ", @reg_select_wait_terms));

30 #####
# Bus-Cycle Status.
#
# One of the major responsibility of the wait-state generator
# is to tell the rest of the world (not the least, the CPU)
35 # where we are in the bus cycle: Beginning, middle, or end.
#
# So. How do we know when a bus cycle begins? The only thing
# we know for sure is that one bus cycle begins when another one
# ends--it's like a Zen koan, isn't it?
40 #
# But seriously. A bus-cycle always ends when there is no longer
# any reason to extend it. Said more prosaically, the signal
# "extend_bus_cycle" will always be zero during the last clock
# of any bus-cycle.
45 #
# So, our only means for deciding that this is the -beginning- of
# a cycle is to notice that "extend_bus_cycle" was false (0).
#
# -- Bonus reason.
50 # Well, there's another way, too, that we can tell if this is
# the start of a bus-cycle: If the CPU (master) is -idle-
# (both read-strobe -and- write strobe inactive). Masters
# sometimes do this. If so, we consider that whole long
# time the "start" of the bus-cycle.
55 #
# **** Reasons To Extend.
#
# There are three "ordinary" reasons to extend a bus-cycle, and
# one "special" one. Let's start with the ordinary (slave_driven) reasons:
60 #
# 1) The peripheral, itself, can request wait-states.
# 2) We can have a fixed (counted) number of wait-states,
# due to both traditional wait-states and setup/hold times.
```

```

#      3) That whole sorry business about registered chip-selects.
#
# And here's the "special" reason
#
5 #      *4) Data has to percolate through the principal databus drive-back
#      register.
#
# But the only way we know when it's time to do (*4) is when
# the bus-cycle is no longer extended for any of the other reasons (1..3).
10 # But (*4) still counts as extension of the current bus-cycle.
# Therefore, we have to account for (1..3) and (4) separately.
# That's OK--I'm an amateur accountant.
#
#
15 # As long as the master is just sittin' there, we take that to
# mean a cycle "is in the process of starting".
&PBM_Wire ("master_is_idle", 1,
    "(. &Get_Sys_Signal($$Sys{master}, "readn" ).) && ".
    "(. &Get_Sys_Signal($$Sys{master}, "writen" ).)" );
20
&PBM_Wire ("slave_wait_asserted", 1, "
    reg_select_wait_asserted ||
    fixed_wait_asserted ||
    periph_wait_asserted ");
25
# The slave gets its chance to wait, or talk, or whatever. Then
# it's over--for good.
# Orion hates &Delay because he never knows which of sync_set or
# sync_reset or any of the other options gets priority.
# reg_slave_phase is defined above because the outgoing chip
30 # selects need to be gated by it.

&Vprint("
always @(posedge clk)
    begin
35         if ((~reset_n) || (bus_cycle_end))
            begin
                slave_phase <= {1 {1'b1}};
            end
            else begin
40                 if (~slave_wait_asserted & ~master_is_idle) begin
                    slave_phase <= {1 {1'b0}};
                end
            end
        end
45     end
    ");

# Don't extend a cycle when the master is idle!
&PBM_Wire ("extend_bus_cycle", 1,
50     "(slave_wait_asserted && slave_phase) ||
    (data_driveback_asserted
    ) ");

# And a bunch of different names for the same thing (used elsewhere):
#
55 &PBM_Assign ("bus_cycle_end", "~extend_bus_cycle && ~master_is_idle");
&PBM_Assign ("bus_cycle_data_ready", "bus_cycle_end" );

# Generate a privately-named version, then assign to global wire:
60 &Delay ("out
    = bus_cycle_start_reg,
    in
    = bus_cycle_end,
    sync_set = master_is_idle,
    set
    = reset,
```

098301061201

```

    ");
    &PBM_Assign ("bus_cycle_start", "bus_cycle_start_reg");

```

```

5  #####
   # Data drive-back timing
   #
   # We are responsible for coming up with this nasty-little signal
   #
10  #      -- data_driveback_asserted.
   #
   # this is true when:
   #
   # 1) This is a read-cycle.
   #
15  # 2) The currently-selected peripheral is -not- on the
   #     "principal" databus
   #
   # 3) There is no longer any other reason to extend the current
20  #     bus cycle (slave_wait_asserted is zero).
   #
   # 4) We're not still gathering-up a data-value as part of a
   #     multiple-bus-cycle dynamic-bus-sizing operation.
   #
25  #     ... and, lest we forget (which we did)...
   #
   # 5) "data_driveback_asserted" wasn't true last time, otherwise
   #     conditions (1)..(4) above will persist forever, and the
   #     master will hang-up indefinitely.
30  #
   # First, come up with a signal that tells us about (2):
   #
   my @driveback_active_terms = ("1'b0");
   if ($$Sys{Principal_Tri_State_Data_Bus}) {
35     foreach $Mod (&Get_Sys_Slave_List ($Sys)) {
         if (($$Mod{Uses_Tri_State_Data_Bus}) &&
             ($$Mod{Tri_State_Data_Bus} eq $$Sys{Principal_Tri_State_Data_Bus}))
         {
             next;
40         }
         push (@driveback_active_terms, $$Mod{internal_active_signal});
     }
   }

45  &Delay ("out = data_driveback_last_time,
          in  = data_driveback_asserted,
          reset = ,
          ");

50  &PBM_Assign ("data_driveback_active", join(" || ", @driveback_active_terms));

   &PBM_Assign ("data_driveback_asserted", "
          data_driveback_active      &&
55          (~$sys_readn)              &&
          (~slave_wait_asserted)     &&
          (~dbs_wait_asserted)       &&
          (~data_driveback_last_time) ");

60  #####
   # Transaction status
   #
   # The wait-state generator is responsible for the transaction-level
   # timing as well as the bus-cycle-level timing. Recall that a transaction

```

09880106-061201

```

# can take multiple bus cycles.
#
# And, once again, we are faced with the same question:  How do
# we know when a transaction starts?  It starts when another
5 # transaction ended (search for "Zen koan," above).
#
# At this time, there's only one way a transaction will keep going
# even when a bus-cycle has ended--if we're in the middle of a
10 # dynamic-bus-sizing operation.
#
&PBM_Wire ("extend_bus_transaction", 1,
          "extend_bus_cycle || dbs_wait_asserted");

# And a bunch of different names for the same thing (used elsewhere):
15 #
&PBM_Wire ("bus_transaction_end", 1, "~extend_bus_transaction");

# Generate a privately-named version, then assign to global wire:
&Delay ("out      = bus_transaction_start_reg,
20       in        = bus_transaction_end,
       set        = reset,
       ");
&PBM_Assign ("bus_transaction_start", "bus_transaction_start_reg");

25 # Lookie here: The --FINAL-- wait-request signal to the master:
#
&PBM_Assign (&Get_Sys_Signal ($Sys{master}, waitrequest),
          "extend_bus_transaction");
}

30 #####
# PBM_Emit_Setup_Hold_Logic
#
# Given a %Sys-hash (reference), emits all the logic to implement
# "custom" readn / writen-strobes.
35 #
# **** Custom Read / Write strobes
#
# If your module requests setup- or hold-times, then it
40 # gets its own private readn- or writen-signal, which is only
# active for a sub-portion of the bus cycle (unlike the masters'
# control signals, which are asserted for the entire cycle).
#
# Each "custom strobe" is generated by an S-R flip-flop.  We
45 # need to assert the strobe at a particular point in the cycle,
# then deassert it at some later point.  We use the counter-value
# (and the load-counter signal) to determine when to assert/deassert
# the custom write-strobe (i.e. when to set/reset the flip-flop).
#
50 # We can only use this scheme because we previously guaranteed that
# all "custom" strobes have at least one clock's worth of setup time.
# This gives us enogh time to compute the S/R inputs to the
# custom-strobe register. (see "Funny setup/hold rule" in
# &Check_Avalon_Rules).
55 #
#####
sub PBM_Emit_Setup_Hold_Logic
(
60   my ($Sys) = (@_);
   &Emit_Comment ("\n Setup- / Hold-Time Logic \n");

#####

```

09880106 "061201
TOTAL 90 "9010886


```

# First, do full-clock setup/hold times.
#
# previous rule-checks guarantee no overlap with fractional case.
# But check anyway.
5 #
foreach $Mod (&Get_Sys_Slave_List ($Sys))
{
    # All the modules we care about will have setup-times.
    next unless $$Mod{Setup_Time} > 0;
10 &Debug (0, "Generating setup/hold-logic for $$Mod{name}");

    $$Mod{Hold_Time} !~ /half/ or die "
        Module $$Mod{name}: internal hold-time consistency check violation";

15 # Come up with an expression that says when to
    # assert the strobe (separate for read- and write-strobes)
    #
    # The strobe is actually asserted on the clock -after- the
    # $read/write_assert_expr goes true, because we have to
20 # propagate through the custom-strobe register.
    #
    my $master_read_expr = "(~".&Get_Sys_Signal($$Sys{master}, "readn").")";
    my $master_write_expr = "(~".&Get_Sys_Signal($$Sys{master}, "writen").")";
    my @read_assert_terms = ($$Mod{internal_active_signal},
25 $master_read_expr);
    my @write_assert_terms = ($$Mod{internal_active_signal},
    $master_write_expr);

    if ($$Mod{Setup_Time} == 1)
    {
30 # Because of the prop-delay (above) we have to order
        # our strobe on the first clock of the bus-cycle. Fortunately,
        # there's a signal just for that purpose:
        #
        push (@read_assert_terms, bus_cycle_start);
        push (@write_assert_terms, bus_cycle_start);
    } else {
        # If we got to here, notice that Setup_Time is 2 or greater.
        # In this case, we have to use the output of the wait counter.
40 # Recall that the wait-counter counts down, so this makes
        # the math a bit of a head-scratcher.
        #
        # Justifying this numerical expression is a bit tricky--
        # I used little text-tables to "simulate" some example
        # count sequences:
45 #
        # Example: Tsu = 2, Th = 0, min_wait-states = 2
        #
        #   wait-counter | expr true? | output write-strobe
        # -----+-----+-----
50 #   --load--         -             idle
        #           1         assert    idle
        #           0         deassert   ACTIVE
        #
        # Example: Tsu = 2, Th = 0, min_wait-states = 5
55 #   wait-counter | expr true? | output write-strobe
        # -----+-----+-----
        #   --load--         -             idle
        #           4         assert    idle
        #           3         -         ACTIVE
60 #           2         -         ACTIVE
        #           1         -         ACTIVE
        #           0         deassert   ACTIVE
        #

```



```

#####
# Now generate narrow write-strobes where a fractional hold-time
# was requested.
5 #
my $has_fractional_hold_times = 0;
# logic-positive "name" for master strobe, to ease understanding:
#
10 my $master_write = "(~".&Get_Sys_Signal($Sys{master}, "writen").")";

my $made_do_shorten_write_strobe = 0;
foreach $Mod (&Get_Sys_Slave_List ($Sys))
{
    next unless $$Mod{Hold_Time} =~ /half/;
15    &Debug (0, "Generating narrow write-strobe for module $$Mod{name}");
    $has_fractional_hold_times++;

    $$Mod{Setup_Time} == 0 or die "
        Module $$Mod{name}: internal hold-time consistency check violation";

    if (!$made_do_shorten_write_strobe)
    {
        &PBM_Wire ("do_shorten_write_strobe"); # forward-declared
        #here.
        $made_do_shorten_write_strobe = 1;
25    }
    &PBM_Assign (&Get_Sys_Signal ($Mod, "writen"),
        "~($master_write && ~do_shorten_write_strobe)");
}

30 if (($has_fractional_hold_times)) {
    &Emit_Comment (" A little dab of shortening for the write-pulse");
    &Delay ("out      = write_second_half,
            in       = $master_write,
            edge    = negedge,
            reset   = reset,
35            ");
    &PBM_Assign ("do_shorten_write_strobe",
        "write_second_half && bus_cycle_end");
40 }
}

#####
# Generate_PBM
45 #
# Given a %Sys-hash (reference), we do everything required to
# generate a PBM, including opening the output-file, blorting
# all logic into it, and subsequently closing the output file
# again.
50 #
#####
sub Generate_PBM
{
    my ($Sys) = (@_);
55    # Open the output file:
    #
    &PBM_Progress ("Creating PBM module: $$Sys{pbm_file}.");
    open (ALTERA_FILEHANDLE2, "> $$Sys{pbm_file}") or die "can't open
60 $$Sys{pbm_file}: $!";
    my $old_out = select(ALTERA_FILEHANDLE2);
    print "$GLOBAL_COPYRIGHT_NOTICE\n";

```

```

&Emit_Module_Header      ($$Sys{pbm_name});

&PBM_Emit_Simple_Assignments ($Sys);
&PBM_Emit_IRQ_Prioritizer   ($Sys);
5  &PBM_Emit_Address_Decoder ($Sys);
&PBM_Emit_Read_Data_Mux    ($Sys);
&PBM_Emit_Dynamic_Bus_Sizer ($Sys);
&PBM_Emit_Wait_State_Generator ($Sys);
10 &PBM_Emit_Setup_Hold_Logic ($Sys);

&Vprint ("\n endmodule // $$Sys{pbm_name}\n");

close (ALTERA_FILEHANDLE2);
select ($old_out);
15 my $synth_list_ref = $$Sys{synth_file_list};
push (@$synth_list_ref, $$Sys{pbm_file});
}

20 #####
# Generate_Core
#
# Given a %Sys-hash (reference), we do everything required to
# generate a system-level "core" module, including opening the output-file,
# blorting all logic into it, and subsequently closing the output file
# again.
#
#####
sub Generate_Core
30 {
    my ($Sys) = (@_);

    # Open the output file:
    #
    35 &PBM_Progress ("Creating Core module: $$Sys{core_file}.");
    open (ALTERA_FILEHANDLE , "> $$Sys{core_file}")
        or die "can't open $$Sys{core_file}: $!";
    my $old_out = select(ALTERA_FILEHANDLE );
    print "$GLOBAL_COPYRIGHT_NOTICE\n";

    40 &Emit_Module_Header      ($$Sys{core_name});
    &Core_Emit_Wire_Declarations ($Sys);
    &Core_Emit_Instances      ($Sys);
    &Vprint ("\n endmodule // $$Sys{core_name}\n");

    45 close (ALTERA_FILEHANDLE);
    select ($old_out);

    my $synth_list_ref = $$Sys{synth_file_list};
    50 push (@$synth_list_ref, $$Sys{core_file});
}

#####
55 # Generate_Wrapper
#
# Given a %Sys-hash (reference), we do everything required to
# generate a system wrapper-module, including opening the output-file,
# writing its contents, and subsequently closing the output file
# again.
60 #
# This function actually does a very simple thing: Create a
# wrapper-module with a single black-box instance in it. It looks
# scary and complicated because it's tri-lingual, and the language

```

09360106 061201

```

# constructs to do this simple thing are more structurally different
# than reason would dictate.
#
#####
5 sub Generate_Wrapper
{
    my ($Sys) = (@_);

    # Open the output file:
10 #
    &PBM_Progress ("Creating Wrapper module: $$Sys(wrapper_file).");
    open (WRAPOUT, "> $$Sys(wrapper_file)") or die
        "can't open $$Sys(wrapper_file): $!";
    my $old_out = select(WRAPOUT);

15 my $core_instance_name = "the_$$Sys(core_name)";

    &Emit_Top_Comment ($$Sys{name}, $$Sys{hdl_language});

20 # AHDL is funny, and requires that we do a semi-C-like "extern"
    # (FUNCTION) declaration of the thing we're about to instantiate:
    #
    &Declare_Ports_For($$Sys{core_name}, 0, $$Sys{hdl_language})
        if $$Sys{hdl_language} =~ /^ahdl/i;

25 &Emit_Module_Header ($$Sys{name}, $$Sys{hdl_language}, 1);

    # Some languages require a little opening dance before the
30 # instantiation:
    if ($$Sys{hdl_language} =~ /^vhdl/i) {
        &Emit_VHDL_Component ($$Sys{core_name});
        &Vprint ("BEGIN\n");

35 } elsif ($$Sys{hdl_language} =~ /^ahdl/i) {
        &Vprint ("VARIABLE\n");
        &Vprint (" $core_instance_name : $$Sys{core_name};\n\n");
        &Vprint ("BEGIN\n");
40 }

    # The instantiation per-se:
    &Instantiate_And_Connect ($$Sys{core_name},
60 $core_instance_name, "", 0,
    $$Sys{hdl_language});

    # And then, of course, everyone requires a little closing-dance:
    if ($$Sys{hdl_language} =~ /^vhdl/i) {
50 &Vprint ("END behavior;\n");

    } elsif ($$Sys{hdl_language} =~ /^ahdl/i) {
        &Vprint ("END;\n");

55 } elsif ($$Sys{hdl_language} =~ /^verilog/i) {
        &Emit_Comment ("\n exemplar attribute $core_instance_name NOOPT TRUE \n");
        &Vprint ("\n // synopsys translate_on \n");
        &Vprint ("\n endmodule // $$Sys{name}\n");
60 }

    close (WRAPOUT);
    select ($old_out);
}

```

```
#####
# Generate_Inc_File
#
# If we're generating an AHDL wrapper, then it's customary to
5 # also make an ".inc"-file. This is, I suppose, the AHDL-equivalent
# of a C-language header (".h") file.
#
# This function generates such a wrapper for the system if
#
10 #####
sub Generate_Inc_File
{
    my ($Sys) = (@_);

15     if ($$Sys{hdl_language} !~ /^ahdl/i) {
        warn ("Suspicious call to Generate_Inc_File.
            (language is $$Sys{hdl_language}, not AHDL)");
        return;
    }

20     $$Sys{inc_file}= "$$Sys{system_directory}/$$Sys{name}.inc";

    &PBM_Progress ("Creating AHDL include-file: $$Sys{inc_file}.");
    open (INCOUT, "> $$Sys{inc_file}") or die
25         "can't open $$Sys{inc_file}: $!";
    my $old_out = select(INCOUT);

    &Emit_Top_Comment ($$Sys{name},    $$Sys{hdl_language});
    &Declare_Ports_For($$Sys{name}, 0, $$Sys{hdl_language});

30     close (INCOUT);
    select ($old_out);
}

35 #####
# Generate_Symbol
#
# Given a ref to the %Sys-hash, this routine generates a Quartus
# BSF-file (schematic symbol) suitable for representing %Sys in a
40 # schematic. The actual work of symbol-layout and geometry is done
# by Aaron Ferrucci's mighty "&Generate_BSF" function in the library
# module "mk_bsf.pm".
#
# We get to say how the ports are arranged by building a list of
45 # port-descriptions. We hand this off to Aaron, and he gives us back
# a string, which is the entire contents of a valid BSF-file (not yet
# created). We then create the file, write Aaron's string into it,
# and we're done.
#
50 #####

####
# Segment-ordering
#
55 # The top-to-bottom order that the symbol-segments is a little
# bit important, and there are many subtle issues, both subjective
# and practical, that come into play. One could write an entire
# program, I'm sure, just fiddling-around with the perfect ordering.
# But it's pretty clear that whatever hash-ordering comes out of
60 # "keys" is probably not what the user wanted.
# I'm going to sort thus:
#
# -- If you're the master, you're first.
```

```

#      -- Any module with "use_tri_state_bus" takes priority.
#      -- alphabetical by class-name
#      -- alphabetical by module name.
#
5 # We're sorting a list of %Mod-refs, so $a and $b are like "$Mod"
#
sub segment_sort
{
10   return -1 if $$a{Is_Bus_Master};
      return 1 if $$b{Is_Bus_Master};

      # No one was the master. The guy with the tri-state databus wins.
      return -1 if $$a{Uses_Tri_State_Data_Bus} && !$$b{Uses_Tri_State_Data_Bus};
      return 1 if !$$a{Uses_Tri_State_Data_Bus} && $$b{Uses_Tri_State_Data_Bus};

15   # It's a tie! Whoever has the alphabetically-first class-name wins:
      my $class_result = $$a{class} cmp $$b{class};
      return $class_result if $class_result != 0;

20   # It's still a tie! Whoever has the alphabetically-first module name wins:
      return $$a{name} cmp $$b{name};
}

sub Generate_Symbol
25 {
      my ($Sys) = (@_);

      &PBM_Progress ("Generating Symbol $$Sys{name}.bsf");

30   # There will -always- be a group at the top with the unwavering
      # clk/reset ports on it.
      #
      my @symbol_segments = ("clk      | 1 | input,
                             reset_n | 1 | input,");

35   # It sure would be nice if there were, say, a database of all the
      # widths and directions of every port on the system-module.
      #
      # HEY! It's our lucky day!
      # Someone already did "&List_Ports_For" on the system-module. Get
40   # handy hashes of directions and widths:
      #
      my $width_hash = &Get_Port_Widths ($$Sys{name});
      my $dir_hash   = &Get_Port_Directions ($$Sys{name});

45   # Put all shared-port groups first:
      foreach $bus_name ($$Sys{tri_state_bus_list}) {
          my @bus_ports = ();
          my $shared_port_table = $$Sys{shared_port_table};
50         foreach $shared_port_name (keys(%$shared_port_table)) {
             my $w = $$width_hash{$shared_port_name};
             my $d = $$dir_hash {$shared_port_name};
             push (@bus_ports, "$shared_port_name | $w | $d");
         }

55         # A null string here should not happen, but don't risk it:
             my $segment_description = join ("\\n", @bus_ports);
             push (@symbol_segments, $segment_description) if $segment_description;
      }

60   # Now go through the system, module-by-module, and make
      # a segment for each one's external (non-shared) ports.
      # Note that we even include the master. Nioses don't have any
      # external ports, but who knows what the future may bring?

```



```

# &Vpp. Because this is sneaky, it's also implementation-dependent
# and error-prone. This will break some day when somebody
# changes the way Vpp works inside, I guarantee it. Global variables
# truly are evil.
5 #
# FUTURE WORK:
#
# I respectfully suggest that the approach used by "generator_functions.vpp"
# could use a good overhaul. Phrasing all that stuff as a real Perl
10 # module would have several advantages--not the least of which would
# be eliminating the need for this kludge you see here.
#
#####
sub Sys_Gen_Housekeeping
15 {
    my ($Sys) = (@_);

    my @Synth_File_List = ();
    $$Sys{synth_file_list} = \@Synth_File_List;

20 $PBM_VERBOSE = $$Sys{verbose}; # More evil globals.

    # run Vpp, but just to "import" the generator-functions library:
    &Vpp ("-Q",
25         "-D",                $$Sys{system_directory},
         "-H", "$$Sys{sopc_directory}/bin/vpp/generator_functions.vpp"
    );

    # Aren't I sneaky?
    # !!BANG!! ... Ouch! My foot!
    #
30 $vpp_pass = 2; # this lets all the vpp-libraries actually print.
}

35 #####
# Generate_PBM_And_System
#
# Given a reference to the system-level PTF section (and
# an %arg-hash that we inherit from mk_systembus), this
40 # function does everything necessary to:
#
# * Create an HDL implementation of the system PBM
# * Create an HDL implementation of the system-level "core" module.
# * Create an HDL wrapper.
45 #
# This task involves rummaging-through the PTF data, opening all the
# appropriately-named files in the appropriate places, writing all the
# HDL-code into them, and tying a bow around the whole mess.
#
50 # Once you've done this, the only thing left for you to do is
# synthesize it and include it in your design.
#
# This function returns (by reference) the constructed database
# known as "%Sys." This way, our caller can snoop on all the
55 # excellent work we've done on his behalf.
#
#####
sub Generate_PBM_And_System
60 {
    my ($arg, $db_Sys) = (@_);

    my %Sys; undef %Sys;
    %Sys = %$arg; # Start off by knowing everything that

```

09880106.061201

mk_systembus knows.

&Sys_Gen_Housekeeping (\%Sys);

5 &Get_System_Data_From_PTF (\%Sys, \$db_Sys);
&Check_Avalon_Rules (\%Sys);
&Create_System_Port_Lists (\%Sys);

10 &Generate_PBM (\%Sys);
&Generate_Core (\%Sys);
&Generate_Wrapper (\%Sys);
&Generate_Inc_File (\%Sys) if \$Sys{hdl_language} =~ /^ahdl/i;
&Generate_Symbol (\%Sys);

15 return (\%Sys); #caller may want to know new info too;
}

20 1; # Perl modules have to say "1".

09980106.061201
T02T90" 90T08860

tables_2D.pm

sub Build_Hash_From_Table

```
{
5   my $table = shift or die "ERROR Build_Hash_From_Table: no table
    specified\n";

    my %Hash;

10   $table =~ s/\#.*$//mg;    #crush comments
    $table =~ s/^\s*\n//mg;    #crush extra new lines
    #$table =~ s/^\s*(.*?)\s*$/$1/mg;    #crush end and begin spaces

    my @line_array = split (/\\n/, $table);

15   #first line is keys for rest of array
    my $old_first_line = shift (@line_array);
    my $first_line = $old_first_line;

20   #convert spaces between words to underscores
    $first_line =~ s/(\\w)\\s+(\\w)/$1\\_$2/g;

    my @key_array = split (/\\|/, $first_line);

25   #second_line is dividing line, check to see no words live there
    my $second_line = shift (@line_array);
    die "Build_Hash_From_Table, second line should be dividing line and
    is not allowed to have words in it"
        if ($second_line =~ /\\w/);

30   foreach $line (@line_array)
    {
        $line .= " "; # add space so splitting \\|\\n gives us the right
                        # size of array.

35   my @tmp_key_array = @key_array;
        my @value_array = split (/\\|/, $line);

        my $value_size = scalar(@value_array);
        my $key_size   = scalar(@key_array);

40   die ("ERROR Build_Hash_From_Table, line\\n($line)($value_size)\\n".
        "splits to a different size than first line\\n".
        "($old_first_line)($key_size)\\n")
50   if ($value_size != $key_size);

        #First column is hash index.
        my $first_hash_index = shift (@value_array);
        $first_hash_index =~ s/^\s*(.*?)\s*$/$1/;
        shift (@tmp_key_array);
        #All other columns are added under hash index
        foreach $value (@value_array)
        {
55   my $name = shift (@tmp_key_array);
            $value =~ s/^\s*(.*?)\s*$/$1/;
            $name =~ s/^\s*(.*?)\s*$/$1/;
            $Hash{$first_hash_index}{$name} = $value;
        }
60   }
    return (%Hash);
}
```

09280106-061201

sub Get_Table_Row_Names

```
{  
    my $hash = shift or die "ERROR Get_Table_Row_Names, no hash specified\n";
```

```
5    die "TYPE MISMATCH ERROR Get_Table_Row_Names, ref hash is  
    (".ref($hash).")\n".  
        "Not (HASH)\n"  
        if (ref($hash) ne "HASH");  
    return (keys %$hash);
```

10 }

sub Get_Table_Column_Names

```
{  
    my $hash = shift or die "ERROR Get_Table_Column_Names, no hash specified\n";
```

```
15    die "TYPE MISMATCH ERROR Get_Table_Column_Names, ref hash is  
    (".ref($hash).")\n".  
        "Not (HASH)\n"  
        if (ref($hash) ne "HASH");
```

20

```
    my @array = &Get_Table_Row_Names($hash);  
    print "debug @array\n";  
    my $row = shift (@array);  
    print "debug $row\n";  
    my @return_array = keys %{$hash->{$row}};  
    print ("DEBUG @return_array DEBUG\n");  
    return (@return_array);
```

25

sub Get_Table_Row_Column

30

```
{  
    my $hash = shift or die "ERROR Get_Table_Row_Column, no hash specified\n";  
    my $row = shift or die "ERROR Get_Table_Row_Column, no Row specified\n";  
    my $column = shift or die "ERROR Get_Table_Row_Column, no Column  
specified\n";
```

35

```
    return ($hash->{$row}{$column});
```

```
}
```

sub Get_Table_XY

40

```
{  
    my ($x,$y) = @_;  
    return (&Get_Table_Row_Column($y,$x));  
}
```

45 sub Table_Row_Column_Is_Defined

```
{  
    my $hash = shift or die "ERROR Table_Row_Column_Is_Defined, no hash  
specified\n";  
    my $row = shift or die "ERROR Table_Row_Column_Is_Defined, no Row  
specified\n";  
    my $column = shift or die "ERROR Table_Row_Column_Is_Defined, no  
Column specified\n";
```

50

```
    return (defined($hash->{$row}{$column}));
```

55 }

sub Table_XY_Is_Defined

```
{
```

60

```
    my ($x,$y) = @_;  
    return (&Table_Row_Column_Is_Defined($y,$x));
```

```
}
```

1;

09030106-051201

v2vhd.pm

#Copyright (C) 2000 Altera Corporation

#####

V2VHD and accompanying functions takes a verilog file and converts it to a
vhd file.
#

#####

This works with most synthesizable verilog code.
Here is what is not supported and known ways to work around it:
0) Verilog numbers must not be more than 32 bits wide
1) No Procedure/Tasks allowed.

2) Asynchronous set/resets must follow clk in always declaration
i.e. always @(posedge reset or posedge clk) not supported,
always @(posedge clk or posedge reset) is.

3) Asynchronous set/resets must be first declared i.e.
always @(posedge clk or posedge reset)

begin
if (reset)
//asynchronous stuff here.
else
//synchronous statements here
endif

4) Params may not define width. i.e. output [Width - 1: 0] foo not supported
5) Params must be able to convert to types of natural or string
6) Parameters must be set using defparam Instance.param=value not #value
Instance

7) Instantiation must connected by Instance (.port (connection)) not
Instance (connection)

8) No case statements allowed. Use a lot of if statements instead.

9) verilog and my code is case sensitive, vhd1 isn't.

#10) don't name a wire/module/instance a vhd1 reserved word.

#11) Integers are forced to be 32 bits wide

#12) verilog /funky_@!#naming_convention_with_slash not supported

#13) timing statements are not supported i.e. a <= #23 b; will be treated as a
<= b;

#14) \$readmemb, \$readmemh and \$write are the only special \$variables supported
#15) It's currently possible to name a wire = tmp_logic later on in the module
or some

other name that get_exclusive name doesn't know about yet. The solution
is to

run through first and gather up all known names then do Exclusive name on
the known name

set

#####

#Ways to make code more readable

1) Put comments back in

2) Find a way to convert boolean to std_logic_vector.

3) Make wire assignments a list with keys = rhs, value = lhs

then when we make a new wire we can look for key rhs before making a new
wire.

4) Find a better way to concatenate /reduce names, sign extend names

5) Fix condition where wire tmp_logical is declared later on after wire name
is

created.

Known Bugs:

Is_real should replace equivalences on parenthesis names.

#####

read_file (\$file_index, \$complete_filename)
#

```
# read_file returns contents of the file named $complete_filename.
# Sounds like a cakewalk except that Verilog has a means of including files
# much like c++ does.
#
5 # So everytime read_file sees:
#
# `include foo
#
# it calls:
10 # &read_file ($file_index+1,foo);
#
# and sticks the result directly in its string, which it returns when it
# reaches the end of its file. (Recursive functions are like that).
#
15 # This file uses a global list called $include_list.
# $include_list is used to ensure that infinite `include loops don't happen
# see check_for_infinite_include_loops
#####
#
20 sub read_file
(
    my ($file_index,$complete_filename,$path) = @_;
    my $file_contents;

25     $complete_filename =~ tr|\\|\/|;
    $path =~ tr|\\|\/|;

    #warn "path,filename $path,$complete_filename\n";
    $path = "." unless $path;
30     open ($file_index,"<$path\/$complete_filename") ||
        die "Unable to open $complete_filename, $!";
    while(<$file_index>)
    {
        if (0)#(s/^s*\`include\s+\\"(.*)\\"//)
35         {
            my $fn = $1;
            #print "in file $complete_filename, found include, $1\n";
            $fn =~ tr|\\|\/|;
            if ($fn !~ /^(\\\\\\)|(\\w\:))/)
40             {
                #full path name isnt specified. Use previous directory location.
                $fn = $path.$fn;
            }

            $include_list{$complete_filename} .= "," if
45 ($include_list{$complete_filename});
            $include_list{$complete_filename} .= $fn;
            #print "include_list for $complete_filename is
            ".$include_list{$complete_filename}.
            #" check value is $complete_filename\n";

50             &check_for_infinite_include_loops($fn,$complete_filename);
            $file_contents .= &read_file($file_index+1,$fn,$path);
        }
        else
55         {
            $file_contents .= $_;
        }
    }
    close($file_index);
60     return ("\\n$file_contents\\n");
}
#####
#
```

```

# check_for_infinite_include_loops
#
# I'm getting sick of recursive functions. Here's another one.
#
5 # This one hunts down the tree structure in $included_file(list) and makes sure
# that files don't refer back on themselves via `includes`.
#
# It is okay for multiple `includes` of the same file as long as that file
# doesn't include something which includes something which includes the original
10 file.
#
# To check, we put a stake in the ground ($check_value) and traverse the
`include` files
# If we ever see our stake again, we know we've walked in a circle.
15 #
#####
#
sub check_for_infinite_include_loops
{
20   my ($included_file,$check_value) = @_;
   #print "c_f_i_i_l, if, $included_file, cf, $check_value\n";
   if (!$include_list{$included_file})
   {
       return;                                     #never heard of this file
25   before
   }                                     #no infinite loop here.

   foreach $key (split(/\./,$include_list{$included_file}))
       # I've seen this guy somewhere before
30   {                                     # make sure I'm not
       circling.
       if ($key eq $check_value)
       {
           die "ERROR: FOUND INFINITE INCLUDE LOOP!\nFILE $check_value
35   EVENTUALLY INCLUDES ITSELF!\n";
       }
       else
       {
           #print "checking $key against $check_value\n";
           #no smoking gun yet, check what this file includes.
           #keep the same $check_value;
           &check_for_infinite_include_loops($key,$check_value);
40       }
       }
   }
45   return;
}

sub Kill_Comments
{
50   my ($line) = @_;

   #Kill multi_line_comments
   my $tmp_line;
   while ($line =~ s|^(.*)\\/(.*)\\*\\/|s)
   {
55       $tmp_line .= $1;
   }
   $tmp_line .= $line;
   #Kill single_line_comments
   $line = "";
60   while ($tmp_line =~ s|^(.*)\\/(.*)\\*\\n|s)
   {
       $line .= $1;
   }
}

```

09880106"051201

```

$line .= $tmp_line;
return ($line);
}
sub Process_Comments
5  (
    my ($line) = @_;
    # Comments
    # VHDL does not have a means for making multi line comments
    # That means that we have to convert the lines to verilog single line
10  comments (//)

    while ($line =~ s|^((.*?)\\/(.*?)\\/(.*)|s)
    {
        my $commented_line = $2;
15        #There better not be nested comments in here!
        die "BAD COMMENT, $commented_line, NESTED COMMENTS!"
            if ($commented_line =~ /\//);

        # Convert all single line comments (//) in the comment to (/--/) so we
20  don't
        # confuse ourselves later.
        while ($commented_line =~ s|\\/(.*)|\\/(.*)|){;}

        $commented_line = join ("\\n\\/", split(/\\n/, $commented_line));

25
        #####
        # Right now we just crush multi-line comments. We could get fancier
        later
        # If you wanted to keep these comments, uncomment the next line.
30        # $line .= "\\n\\/$commented_line\\n"
    }

    #####
    # Now we just have single comments left. For these, we can just convert
35    # the verilog single line comment // to the vhd1 single line comment --.
    # We also need to convert all ticked statements, e.g. (`define, `ifdef,
    `endif, etc.)
    # to their non-ticked counterparts, (define, ifdef, endif, etc.) And since
40  we
    # later split commands on semi-colon(;) Crush ; so that we won't have to
    worry about it.

    # crush single-line comments
    while ($line =~ s|^((.*?)\\/(.*?)\\n(.*)|s)
45    {
        my $comment = $2;
        my $rest = $3;
        #warn "found single line comment named $comment, $1\\n";
        #Keep comments with special word "exemplar" in them for Leonardo
50  Spectrum.
        #But convert comment character to vhd1 comment character so that we don't
        #get stuck in an infinite loop.

        if ($comment =~ /^s*exemplar/i)
55        {
            #warn "putting $comment back into line\\n";
            $line .= "\\-\\-$comment\\n";
        }
        $line .= $rest;

60
        #####
        # Just kill comments for now
        # while ($comment =~ s/(\\`|\\;)//){;}

```

TOP SECRET 901061000000


```

    # $line .= "\-\\-$comment";
}

return($line);
5 }

#####
#
# Process_Verilog_Directives takes verilog code as its sole string argument.
10 # It then processs all things in the file that have ` associated with it except
# for `include
# such ` thingees include
# `define foo 3
# `ifdef, `else, `endif
15 # `foo
# It returns an equivalent verilog string devoid of all `marks. It replaces
# all `foo with its definition if defined. (No error message is printed if foo
# has not been defined). It does the correct thing on `ifdef, `else, `endif
# statements, (including nested ifdefs).
20 #####
#
sub Process_Verilog_Directives
{
    my ($line) = @_;
    my $defined_line = "";

    undef %defined;
    undef %definition_of;
    my %defined;
    my %definition_of;
    my @nested_ifdefs = (1); #start off including code.

    my $I_Should_Keep_This_Part;

35 $line = " ".$line; #a space is added so that first time through, split
//, we pass through
#all of the gates and emerged unscathed as the first part of $defined_line;

#print "line is $line\n";
40 foreach $tick (split (/\/s,$line))
{
    $I_Should_Keep_This_Part = eval (join ('&&', @nested_ifdefs));
    #print ("tick is $tick, isktip is $I_Should_Keep_This_Part, nifdef = "
    #.join ('|',@nested_ifdefs). "\nundefined line now is $defined_line ----
45 \n");
    die "unmatched `endif" if ((scalar (@nested_ifdefs)) == 0);
    if ($tick =~ /^ifdef\s+(\S+)(.*)/s)
    {
        $defined{$1} = 0 unless ($defined{$1});
        $I_Should_Keep_This_Part = ($defined{$1} &&
50 $I_Should_Keep_This_Part);
        push (@nested_ifdefs,$I_Should_Keep_This_Part);
        $defined_line .= $2 if ($I_Should_Keep_This_Part);
        next;
55 }

    if ($tick =~ /^else\s+(.*)/s)
    {
        die ("saw `else before `ifdef") unless ((scalar (@nested_ifdefs) >
60 1));
        $I_Should_Keep_This_Part = pop (@nested_ifdefs);
        $I_Should_Keep_This_Part = (! $I_Should_Keep_This_Part) && eval (join
('&&', @nested_ifdefs));
    }
}

```

09201061201
 1021901901061201

090830106 "061201

```
        push (@nested_ifdefs,$I_Should_Keep_This_Part);
        $defined_line .= $1 if ($I_Should_Keep_This_Part);
        next;
    }
5
    if ($tick =~ /^endif\s+(.*)/s)
    {
        die ("saw `endif` before `ifdef` in $line\n") unless ((scalar
10      (@nested_ifdefs) > 1));
        pop (@nested_ifdefs);
        $I_Should_Keep_This_Part = eval (join ('&&', @nested_ifdefs));
        #print "endif says isktip is $I_Should_Keep_This_Part,nif ".join
        ('|',@nested_ifdefs);
        $defined_line .= $1 if ($I_Should_Keep_This_Part);
15      next;
    }

    if ($tick =~ /\Adefine\s+(\S+)\s*(\S*)\s*$/m)
    {
20      if ($I_Should_Keep_This_Part)
      {
          #print "definition of $1 is $2, vpp pass is $vpp_pass";
          #die "$1 is defined in 2 places" if ($defined{$1});
          $defined{$1} = 1;
          $definition_of{$1} = $2;
          $tick =~ s/^(.*)\n(.*)/$2/s;
          $defined_line .= $tick;
      }
      next;
30    }

    if ($tick =~ /^(\S+)(.*)/s)
    {
        $tick = $definition_of{$1}.$2;
        $defined_line .= $tick if ($I_Should_Keep_This_Part);
        next;
35    }

    if ($tick =~ /^s+/s)
    {
        #this should only be the first one.
        $defined_line .= $tick if ($I_Should_Keep_This_Part);
        next;
40    }
}
45
die "missing `endif`" if ((scalar (@nested_ifdefs)) > 1);

return ($defined_line);
}
50
sub date_time
{
    my ($sec,$min,$hour,$mday,$mon,$year,$yday,$isdet) = localtime(time);
    $mon++;
55    $year += 1900;

    my $d = sprintf("%04d.%02d.%02d",$year,$mon,$mday);
    my $t = sprintf("%02d:%02d:%02d",$hour,$min,$sec);

60    return "$d $t";
}
```

```

#####
#
# Verilog_Number_To_Bit_String
#
5 # Verilog_Number_To_Bit_String takes a verilog number of the form 5'hA
# and turns it into a quoted bit string "01010".
#####
#
10 sub Verilog_Number_To_Bit_String
(
    my ($verilog_number) = @_;
    my $integer_value = 0;
    my $width;

15     $verilog_number =~ s/^\s*(.*?)\s*$/$1/;

    die "ERROR NO WIDTH SPECIFIED FOR VERILOG NUMBER $verilog_number\n"
        unless ($verilog_number =~ s/^\s*(\d+)\s*$//);

20     $width = $1;
    die "width is greater than 32 bits. I intend to fix this, but for now, you
are out of luck\n"
        if ($width > 32);
    # If there is an 'x' or 'z' in the number, return
    # a bitstream of x or z with width $width

25     if ($verilog_number =~ /[zx]/i)
    {
        my $tmp_string = $1 x $width;
        return ("\"$tmp_string\"");
    }

30     if ($verilog_number =~ /^b([0-1]+)$/i)
    {
        foreach $bit (split (//,$1))
        {
            $integer_value = $integer_value * 2;
            $integer_value += 1 if ($bit == 1);
        }

40     }

    if ($verilog_number =~ /^d([0-9]+)$/i)
    {
        $integer_value = $1;

45     }

    if ($verilog_number =~ /^o([0-7]+)$/i)
    {
        $integer_value = eval("0".$1);

50     }

    if ($verilog_number =~ /^h([0-9a-f]+)$/i)
    {
        $integer_value = eval("0x".$1);

55     }

    die ("ERROR Verilog_Number_To_Bit_Size_And_Bit_String:\n"
        . "NUMBER $verilog_number NOT UNDERSTOOD!")
        if ($integer_value eq "");

60     if ($integer_value >= 2**32)
    {

```

```

        #warn " verilog_number $verilog_number int_value ($integer_value) is
bigger than 2**32\n";
        my @bit_array;
        while ($integer_value > 0)
5          {
            unshift (@bit_array, ($integer_value % 2));
            $integer_value = $integer_value >> 1;
          }
        my $bit_string = join ("", @bit_array);
10        #warn "test way to do it yields $bit_string\n";
      }

      #Orion, warn if number is >= than 2>>$width

15      my $return_string;
      my $mask_value;
      foreach $mask_bit (reverse(0..($width-1)))
      {
        if ($integer_value & (1 << $mask_bit))
20          {
            $return_string .= "1" ;
          }
        else
        {
25          $return_string .= "0" ;
        }
      }
      #return ("\'$return_string\'") if ($width == 1);
      return ("\"$return_string\"");
30    }

    sub VN2BS {return ( &Verilog_Number_To_Bit_String(@_));}

    sub Process_Concatenation
35    {
      my ($string,
        @Module_Info) = @_;
      my (
        $Width_List,
        $Signal_List,
40        $pWire_Assignments,
        $Equivalence_List) = @Module_Info;

      my ($begin, $middle, $end) = &Count_Parentheses($string, '\{, '\}');
45      #warn "PC: middle ($middle)\n";
      #warn %$Width_List;
      #warn "PC: end\n";
      die "ERROR Process_Concatenation, NO VALUE BETWEEN SQUIGGLY BRACKETS in
($string)\n"
50      if ($middle eq "");
      while ($middle =~ s/\{|\}\//g){;} #e.g. {a,b,c,{a,e},f,g} -> {a,b,c,a,e,f,g}

      my ($expanded_array, $width) =
&Expand_Array_Of_Bit_Vectors_Into_Separated_Bits(" ", $middle, @Module_Info);
55      #warn "P_C, return_width is $return_width, width is $width\n";
      $string = "$begin";
      if (scalar(split (/\.\/, $expanded_array)) == 1)
      {
60        my $tmp_string = $expanded_array;
        while ($tmp_string =~ s/\'/\'"/){;}

        #Turn indexed bits into bit array of size 1
        while ($tmp_string =~ s/\[s*(\d+)\s*\]/\[1:$1\]/s){;}

```

00880106-061201

```

    $string .= $tmp_string;
}
else
{
5    $string .= " std_logic_vector\' ";

    $string .= "(";
    $string .= $expanded_array;
    $string .= " \)$end";
10 }
#warn "PC: Concatenation = $string\n";
return (&Replace ("Concatenation = $string",$width,@Module_Info));
}

15 #####
#
# Expand_Array_Of_Bit_Vectors_Into_Separated_Bits
#
# Does exactly what it says. e.g.
20 # &Expand_Array_Of_Bit_Vectors_Into_Separated_Bits (A(3 DOWNT0 2),"0010")
# returns "A[3],A[2],\'0\',\'0\',\'1\',\'0\'"
# We convert brackets to VHDL Parentheses at the end of the module
#####
#
25 sub Expand_Array_Of_Bit_Vectors_Into_Separated_Bits
{
    my ($separator,
30     $Comma_Separated_String,
    @Module_Info) = @_;

    my ($Width_List,
        $Signal_List,
        $pWire_Assignments,
35     $Equivalence_List) = @Module_Info;

    my $string = "";
    my $width = 0;

40     foreach $name (split(/\s*\./,\s*/s,$Comma_Separated_String))
    {
        $name =~ s/^\s*(.*?)\s*$/\1/s;
        #warn "eabvisb name ($name)\n";

45     $name = &V2VHD_Equation($name, @Module_Info);

        #Convert Bit String into bits, i.e. "10110" => \'1\',\'0\',\'1\',\'1\',\'0\'
        if (&Replace_Equivalences($name,$Equivalence_List) ==~
50     /\^(\"|\')([01XZ]+)\1$/i)
        {
            foreach $bit (split(/\./,$2))
            {
                $string .= "$separator " if ($string);
                $string .= "\'$bit\'";
                $width++;
                #warn " eabvisb bit_string bit is $bit, width is $width\n";
            }
        }
        else
60     {
            my ($left,$right);

```

09080106:061201

09280106.061201
TOT190.90T08860

```
($name,$left,$right) = &Vector_Range($name,$Width_List);

if ($left eq "") {$left = &Width_Of($name,$Width_List);}
if ($right eq "")
5   {
    $name = &Add_Intermediate_Signal("tmp_$name" =
$name", $left, @Module_Info);
    $left = eval ($left - 1);
    $right = 0;
10   }

    $left = eval($left); $right = eval($right);
    foreach $index (&Order($left,$right))
    {
15      $width++;
      $string .= "$separator\n\t" if ($string);
      $string .= "$name\[ $index\]";
      #warn " eaobvisb ($name($index)$separator) added to string, width
20      is $width \n";
    }
  }
}

return($string,$width);
25 }

#####
#
# Vector_Order
30 # Inputs, $left, $right
#
# Vector_Order (0,4) returns (0 TO 4)
# Vector_Order (3,1) returns (3 DOWNTO 0)
#####
35 #

sub Vector_Order
{
40   my ($left_index,$right_index) = @_;
   return "($left_index)"
   if ($right_index eq "");
   if (($left_index >= $right_index) || ($right_index == 0))
   {
45     return "($left_index DOWNTO $right_index)";
   }
   else
   {
     return "($left_index TO $right_index)";
50   }
}

#####
#
# Order
55 # Inputs, $left, $right
#
# Order (0,4) returns array (0,1,2,3,4)
# Order (3,1) returns array (3,2,1)
#####
60 #

sub Order
{
  my ($left,$right) = @_;
```

```

my @Vector_Array;
die "ERROR Order: LEFT VALUE ($left) NOT A NUMBER\n"
    unless ($left =~ s/^\s*(\d+)\s*$/$1/s);
die "ERROR Order: RIGHT VALUE ($right) NOT A NUMBER\n"
    unless ($right =~ s/^\s*(\d+)\s*$/$1/s);

if ($right > $left){@Vector_Array = ($left..$right);}
else{ @Vector_Array = reverse ($right..$left);}
return (@Vector_Array);
}

#####
# Width_Of
#
# Returns the width of a variable, verilog number, or vhd1 bit_stream.
# eg. Width_Of(4'h2) = 4; Width_Of "01010" = 5;
# Returns "" if the width is not known.
#
#####
sub Width_Of
{
    my ($var, $Width_List, $pParameter) = @_;

    #KILL SPACES! KILL! KILL!
    $var =~ s/^\s*(.*?)\s*$/$1/s;

    #A verilog number is nice enough to tell you its width
    #at the very beginning. Usually a pain, but in this case it is great
    if ($var =~ /^(\d+)\'[bdoh]([\d+a-fxz])/i)
    {
        #warn " WIDTH_OF found verilog number $var, returned width of $1\n";
        return ($1);
    }

    #Count the bits in a vhd1 bit stream
    if ($var =~ /^(\\"|\\')([01XZ]+)\1/i)
    {
        my $width = scalar (split (//,$2));
        #warn " WIDTH_OF found vhd1 number $var, returned width of $width\n";
        return ($width);
    }

    my ($name,$left,$right) = &Vector_Range($var, $Width_List, $pParameter);
    #warn "Width_Of $var after Vector_Range, left,right -> $left, $right\n";
    return "" if ($left eq ""); #Not Known;
    return "$left" if ($right eq ""); #Known, but not a vector.
    return (abs($left - $right) + 1); #vector width arithmetic
}

#####
# Vector_Range
#
# Returns the name and (left and right value) of the vector range for a
# verilog or vhd1 vector.
# if
# reg [4:0] foo;
# &Vector_Range (foo [3:2]) = (foo,3,2)
# &Vector_Range (foo) = (foo,4,0)
# &Vector_Range (foo[2]) = (foo,2,2)
#####
sub Vector_Range
{
    my ($var, $Width_List, $pParameter) = @_;
```

```

my $name;
my $left;
my $right;

```

```

5   $var =~ s/^\s*(.*?)\s*$/s;

```

```

#If our var is foo [3:2] return (foo,3,2)
#We're not sophisticated enough to deal with (>1)-dimensional
#vectors. If you need a multi-dimensional vector width,
10  #there is a keyboard in front of you, start typing.

```

```

if ($var =~ /(\w+)\s*\[(.*?)\]/s)
{
    #warn " Width_Of brackets, inside brackets is $2\n";
15  $name = $1;
    ($left,$right) = split (/s*:\s*/s,$2);
    $left = eval($left);
    $right = $left if ($right eq "");
    $right = eval($right);
20  return ($name,$left,$right);
}

```

```

#If our var is foo (3 DOWNT0 2) return (foo,3,2)
if ($var =~ /(\w+)\s*\((.*?)\)/s)
{
    #warn " Width_Of brackets, inside brackets is $2\n";
    $name = $1;
    my $index = $2;
    ($left,$right) = split (/s*(DOWN)?TO\s*/s,$index);
30  $left = eval($left);
    $right = $left if ($right eq "");
    $right = eval($right);
    return ($name,$left,$right);
}

```

```

#No indecies are specified, that means use the whole
#variable. Fortunately, we have already saved all
#variable widths in %$Width_List
40  ($left,$right) = split (/\/,/, $$Width_List{$var});
    $name = $var;
    return ($name, $left,$right);
}

```

```

#####
45  #
    # Replace_Parameters
    # Replaces all text in $val with $$pParameter{text}
    # Currently not being used

```

```

50  sub Replace_Parameters
    {

```

```

        my ($val, $pParameter) = @_;
        while ($val =~ s/^(.*?) (\b[a-zA-Z_\]\w*) (.*?)$/s)
        {
55          last if ($2 =~ /integer/i);
          my $param_substitution = $$pParameter{$2};
          die "ERROR PARAMETER SUBSTITUTION: PARAMETER $2 NOT DEFINED\n"
              if ( $param_substitution eq "");
          $val .= $param_substitution.$3;
60        }
        return (eval ($val));
    }
}

```

09880106-061201


```

#####
#
# Get_Port_Name_Direction_And_Type
# Takes a vhdl port string and returns Name, Direction, Type and Vector Range
5 # i.e. SIGNAL data_from_cpu : OUT STD_LOGIC_VECTOR
# returns ("data_from_cpu","OUT","STD_LOGIC_VECTOR(31 DOWNT0 0)")
#
sub Get_Port_Name_Direction_And_Type
{
10   my ($string) = @_;
   my $possible_variables = 'SIGNAL|VARIABLE|SHARED\s+VARIABLE';
   if ($string =~ /^\\s*($possible_variables)\\s+(\\w+)\\s*:\\s*(\\w+)\\s*(.*)$/is)
   {
15     my ($name,$direction,$type) = ($2,$3,$4);
     return ($name,$direction,$type);
   }
   else
   {
20     die "ERROR Get_Port_Name_Direction_And_Type, IMPROPER STRING $string\\n";
   }
}

#####
#
25 # Declare_Signal
#
# takes, $name,$pSignal_Width as arguments, returns
# SIGNAL $name : STD_LOGIC_VECTOR ($left DOWNT0 $right)
# or
30 # SIGNAL $name : STD_LOGIC_VECTOR ($left TO $right)
# depending on pSignal_Width

sub Declare_Signal
{
35   my ($name,$pSignal_Width) = @_;
   my ($left,$right) = split (/\\/,,$$pSignal_Width{$name});
   die "Declare_Signal, bad inputs ($name,$left,$right)\\n"
     if (($name =~ /\\W/) || ($left =~ /\\D/) || ($right =~ /\\D/));
   my $Signal_Declaration;
40   if ($left < $right)
   {
     $Signal_Declaration = "SIGNAL $name : STD_LOGIC_VECTOR ($left TO
$right);";
   }
45   else
   {
     $Signal_Declaration = "SIGNAL $name : STD_LOGIC_VECTOR ($left DOWNT0
$right);";
   }
50   #warn "DECLARE_SIGNAL RETURNINGG $Signal_Declaration\\n";
   return ($Signal_Declaration);
}

#####
#
55 # Convert_Signals_To_Shared_Variable
#
# Takes a string and a pointer to Signal_List.
# Converts every word(\\b([a-zA-Z]\\w*)\\b) in the Signal_List from a SIGNAL
# to a SHARED VARIABLE. Returns an array of all words changed.
60 # This function is only needed to handle blocking statements.

sub Convert_Signals_To_Shared_Variable
{

```

```

my ($line,$Signal_List) = @_;
die "ERROR Convert_Signals_To_Shared_Variable, Signal_List IS NULL\n"
    if (!$Signal_List);
my @return_array;
5 while ($line =~ s/\b([a-zA-Z]\w*)\b//)
    {
        my $variable = $1;
        #warn "variable is $variable, line is ($line)\n";
        push (@return_array,$variable);
10 my $tmp_Signal_List = $$Signal_List{$variable};
        die "ERROR VARIABLE CONVERSION: of ($variable) FAILED
($tmp_Signal_List)\n"
            unless ($tmp_Signal_List =~ s/^(\\s*)(SIGNAL|SHARED VARIABLE)/$1SHARED
VARIABLE/s);
15 $$Signal_List{$variable} = $tmp_Signal_List;
    }
}

sub Process_Register_Assignment
20 {
    my ($line,@Module_Info) = @_;
    my ($Width_List,
        $Signal_List,
        $pWire_Assignments) = @Module_Info;
25 my $lhs;
    my $rhs;
    my $lhs_width;
    my $after_line;

30 #Process $readmem(b|h), $write, etc
    #warn "line is $line\n";
    if ($line =~ s/^(\\s*)(\\$.??)\\s*/$2/s)
    {
35 return ($1.&Process_Dollar_Verilog_Statements($line,@Module_Info));
    }

    #Search for delay patterns
    $line =~ s/(.*)\\#\\s*\\S*(.*)/$1 $2/s;
    # "#" delay kind of broken due to differences between languages
    # representation of delays, so just forget about them for now
    if (0)if ($line =~ s/(.*)\\#\\s*(.*)/$1/s)
    {
40 my $wait_amount = $2;
        my ($tmp,$delay_amount,$rest_of_line);

45 if ($wait_amount =~ /^\\(\\/)
        {
            ($tmp,$wait_amount,$rest_of_line) = &Count_Parentheses($wait_amount);
50 $line .= $rest_of_line;
        }
        else
        {
            $wait_amount =~ s/^(\\S+)(.*)/$1/s;
            $line .= $beginning_of_line.$2;
55 }
        #If there is only a delay statement, e.g. (#32;) return "wait for 32 ns;"
        if ($line =~ /^\\s*\\;/s)
        {
60 return ("WAIT FOR $wait_amount ns;");
        }
        else
        {
            $after_line = "AFTER $wait_amount ns";

```

```

    }
}

5   #There are two kinds of verilog assignments that can be made
    # 1) blocking "=" assignment made immediately
    # 2) non-blocking "<=" assignment made after all other items in that
    #     timestamp are equated

10  if ($line =~ /^(.*?)(\<|=|\\=)(.*)$/s)
    {
        my ($lhs,$operator,$rhs) = ($1,$2,$3);
        my $tmp_pWire_Assignments;
        my $line = &V2VHD_Equation_Wrapper (" $lhs =
15  $rhs", $Width_List, $Signal_List, \ $tmp_pWire_Assignments);
        #warn "P_R_A: After Wrapper wire assignments are $pWire_Assignments\n";

        if ($operator eq "\\=")
        {
20          $line =~ s/^(.*?)\<(\=.*?)/$1:$2/s;
          #warn "found blocking statement ($lhs$operator$rhs) lhs is ($lhs)\n";

          #####
          # BLOCKING STATEMENTS are tricky.
          # Verilog has this concept of "blocking and non-blocking" statements.
          # VHDL has module scoped SIGNALS that can only be non-blocking and
          # process scoped VARIABLES that must be blocking. To convert
          # verilog blocking statements into an equivalent VHDL blocking
          statement,
30          # we must first convert the lhs of the blocking assignments within
          our process
          # into VARIABLES. At the end of our process, we transform all our
          VARIABLE
          # names into temporary variable names and assign the original SIGNAL
          names
35          # equal to the temporary variable names. Perhaps an example will
          keep my
          # confusing explanation from spinning your head any longer.
          #
          # always @(posedge clk)
          #   a = a + 1;
          #
          # will become:
          # PROCESS BEGIN
45          # WAIT UNTIL clk = "1";
          #
          # number_1_1_bits_wide := "1";
          --all machine generated equations
          #   a_2_bits_wide := std_logic_vector' ('0', variable_a(0)) +
          --must precede assignment and be
          #
          # std_logic_vector' ('0',
50  number_1_1_bits_wide(0)); --of type VARIABLE
          #   variable_a := a_2_bits_wide(0 DOWNT0 0);
          #   a <= variable_a;
          # END PROCESS
          #
55          #####

          #####
          # First, we convert any machine generated wire assignments to
60  variable
          # assignments within the line since they need to be evaluated before
          # the "effective immediate"(ly) variable assignment.
          # in the above example, number_1_1_bits_wide and a_2_bits_wide are

```

09880106-061201

```
# "machine generated".

my $machine_generated_commands = "";
#warn "tmp_WA ($tmp_pWire_Assignments)\n";
5 my (@blocking_commands) = split (/\/, $tmp_pWire_Assignments);
foreach $block_command (@blocking_commands)
{
    $block_command =~ s/^(.*?)\<(\=.*?)$/\$1:\$2;/s;
    &Convert_Signals_To_Shared_Variable($1, $Signal_List);
10 $machine_generated_commands .= "$block_command";
}
$line = $machine_generated_commands.$line;

#####
15 # Now we'd like to translate all left hand side arguments in the
process # into tmp names. Unfortunately, we only know what's in this line,
not # what is in the entire process. So for each chip design, we put a
20 non-vhdl # sentence that says "Please Convert $signal_name To A Variable" in
$line. # We'll do the conversion later.

25 while ($lhs =~ s/\b([a-zA-Z]\w*)\b/)
{
    $line .= "Please Convert $1 To A Variable";
}

30 }
else {$$pWire_Assignments .= $tmp_pWire_Assignments;}

if ($after_line ne "")
{
    35 $line =~ s/\;\s*$/;/s;
    return ("$line $after_line;");
}
else
{
    40 return ("$line");
}
}
else
{
    45 return "";
}
}

sub Get_Exclusive_VHDL_Name
50 {
    my ($name, $List) = @_ ;

    $name =~ tr/A-Z/a-z/;
    #First make $name into a vhdl acceptable name
    55 #warn "eatme 2, $name\n"
    #if ($GLOBAL_DEBUG);
    #Say goodbye to everything that is not a word character in name.
    while ($name =~ s/\W+//){;}

    60 #Convert spaces to underscores
    while ($name =~ s/\s+/\_/s){;}
}
```

```

#But not too many underscores, vhdl will not let you have two underscores
in your name or
#any underscores at either end
while ($name =~ s/(\_){2,}/\_s/){;}

5
while ($name =~ s/^\_+([.*?])\_+$/\1/){;}

#vhdl does not like names to begin with numbers
$name =~ s/^\(d)/number_\1/;
10
#warn "eatme\n";
#warn "IN GETXNAME\n"
#if ($GLOBAL_DEBUG);
while ($$List{$name} ne "")
{
15
    #warn "G_E_N name $name already taken. Trying tmp_$name\n"
    #if ($GLOBAL_DEBUG);
    $name = "tmp_$name";
}
#warn "G_E_N name $name not taken.\n"
20
#if ($GLOBAL_DEBUG);
#Keep Place holder at this name.
$$List{$name} = "Taken";
return ($name);
}
#####
#
# Convert_Integers_To_Std_Logic_Vector
#
# Convert_Integers_To_Std_Logic_Vector takes left, operator, right as inputs.
30
# If left/right are integers, it converts them to a bit vector.
# If the widths of left or right is not known, it sets the unknown width equal
to the known width.
#
# (Perhaps it should not do this?)
#
35
# It does not use the operator now, but it maybe useful later.

sub Convert_Integers_To_Std_Logic_Vector
{
40
    my ($left,$operator,$right,$Width_List,$Equivalence_List) = @_;
    my $left_width = &Width_Of($left,$Width_List);
    my $right_width = &Width_Of($right,$Width_List);

    #Check that widths are equal if both are known.
    if (($left_width) && ($right_width))
45
    {
        if ($left_width != $right_width)
        {
            my $tmp_left = &Replace_Equivalences ($left,$Equivalence_List);
            my $tmp_right = &Replace_Equivalences ($right,$Equivalence_List);
50
            #warn ("WARNING EXPRESSION ($left $operator $right)\n"
            #."WIDTH OF $tmp_left ($left_width) != WIDTH OF $tmp_right
($right_width)!\n");
        }
    }
55
    else
        #Warn if neither width is known.
        {
            if (!($left_width || $right_width))
            {
60
                die ("CV2SLV VECTOR WIDTHS ($left_width) ($right_width) UNKNOWN FOR
EXPRESSION ARITHMETIC OPERATION ($2) ($3) ($4)\n");
            }
            else
                #If only one width is known, convert the unknown width.

```

```

    {
        if ($left_width) #right width is unknown.
        {
            if ($right =~ /\s*(\d+)\s*$/s) #number => verilog decimal number
5      with width $left_width
            {
                #warn ("right width is unknown, left width is $left_width\n".
                #"sending $left_width\`d$1 to VN2BS");
                $right = &VN2BS("$left_width\`d$1");
10          }
            }
        else #left width is unknown.
        {
            if ($left =~ /\s*(\d+)\s*$/s) #number => verilog decimal number
15      with width $right_width
            {
                #warn ("left width is unknown, right width is $right_width\n".
                #"sending $right_width\`d$1 to VN2BS");
                $left = &VN2BS("$right_width\`d$1");
20          }
            }
        }
    }
    #Now determine the proper width.
    #take max
    my $max_width = $right_width;
    $max_width = $left_width
        if ($left_width > $right_width);
30    return ($max_width,$left,$right);
}

#####
#
# Replace_Equivalences
# Replaces all tmp names in Equivalence List. Since some names
# In Equivalence_List refer to other names in the same Equivalent_List,
# Replace_Equivalences is Recursive. See big comment before sub V2VHD_Equation
# for an explanation of the bigger picture.
40 sub Replace_Equivalences
    {
        my ($string,$Equivalence_List) = @_;
        my $new_string;
        my $replacement_value;
        while ($string ne "")
        {
            if ($string =~ s/^\([\W\`\d]+\)/s){$new_string .= $1;next;}
            if ($string =~ s/^\(\w+\)/s)
50          {
                my $word = $1;
                $replacement_value = $$Equivalence_List{$word};
                if ($replacement_value eq "")
                {
55                  $new_string .= $word;
                }
                else
                {
                    my $new_replacement_value
60      &Replace_Equivalences($replacement_value,
                                $Equivalence_List);

                }
            }
        }
    }
    #####

```

```

#parentheses get replaced if there is additional replacement levels
#inside the parentheses. e.g. verilog statement
# (a+b) => (binary) => parentheses.
# When replacing,
5 # parentheses => (binary) => (a+b)
#
# (a) => parentheses
# When replacing
# parentheses => a //no parentheses
10
    if ( ($word =~ /parentheses/i) &&
($$Equivalence_List{$replacement_value}))
    {
        $new_replacement_value = "\($new_replacement_value\)";
15     }
    $new_string .= $new_replacement_value;
    }
}
20 return ($new_string);
}

sub Vestigual_Replace_Equivalences
(
25 my ($string,$tmp_Equivalence_List) = @_ ;
my %Equivalence_List = %$tmp_Equivalence_List;

my $values_were_changed = 1;
while ($values_were_changed)
30 {
    $values_were_changed = 0;
    foreach $key (keys(%$Equivalence_List))
    {
        my $value = $$Equivalence_List{$key};
35
        #Put parentheses around value if key was parenthesized
        #And it needs it
        if ( ($key =~ /Parentheses/) && ($$Equivalence_List{$value}))
        { $value = "\($value\)"; }
40
        while ($string =~ s/\b$key\b/$value/)
        {
            #warn "key is $key\n value is $value\n string is $string\n";
            $values_were_changed = 1;
            undef $$Equivalence_List{$key};
45        }
    }
}

#Now crush parentheses around single objects
50 #while ($string =~ s/\(((([\sa-zA-z\"' ][\s\\w\"' ]*)\\)/$1/s){;})

#warn "Replace_Equivalences, done, string is $string\n";
return ($string);
55 }

#####
#
# ReadMem takes a verilog $readmem(b|h)(<filename>,<memory>)
# instruction and then hardwires _memory_ to the values in _filename_.
60 # This isn't as flexible as a true conversion of readmem would be (<filename>
# could be parameterizable and be different for multiple instances of the
# same module. In the future, this should be replaced with a real vhd1

```

09880106 061201
T02190-00T0880

```
# equivalent, but it is so much easier to parse filenames with perl than with
vhdl
```

```
sub ReadMem
```

```
5  {
    my ($mem_radix,
        $dat_filename,
        $mem_name,
        @Module_Info) = @_;

10     my ($Width_List,
        $Signal_List,
        $pWire_Assignments,
        $Equivalence_List) = @Module_Info;

15     my $return_string;
    my ($left,$right,$up,$down) = split (/s*\,\s*/s,$Width_List{$mem_name});
    my ($width) = &Width_Of($mem_name,$Width_List);
    open (DAT,"< $dat_filename") or
20     open (DAT,"< nios_system_sim\/$dat_filename") or
        die "Cannot open $dat_filename ($!)\n";
    my $dat_contents;
    while (<DAT>)
    {
25         if (s/^(.*?)\\\/.*$/1/){;}
        $dat_contents .= $_
            unless (/^\s*$/s);
    }
    close (DAT);

30     my @Address_Data_Pairs = split(/\/@/, $dat_contents);
    foreach $address_data (@Address_Data_Pairs)
    {
35         my ($address,@data) = split (/s+/s,$address_data);
        #warn "address is $address. data is @data\n";
        my $integer_address = eval ("0x$address");
        foreach $datum (@data)
        {
40             if ($mem_radix eq "b")
            {
                $datum = "\"$datum\"";
            }
            else
45             {
                $datum = &VN2BS("$width\'$mem_radix$datum");
            }

            $return_string .= "$mem_name($integer_address) <= $datum;\n";
50             $integer_address++;
        }
    }
    return ($return_string);
55 }

#####
#
# Process_Dollar_Verilog_Statements
#
60 # Converts verilog $readmemb, $readmemh and $write statements
# to the appropriate vhdl equivalent
#
sub Process_Dollar_Verilog_Statements
```

09080106 061201


```

{
    my ($equation,@Module_Info) = @_;

    my ($Width_List,
5      $Signal_List,
      $pWire_Assignments,
      ) = @Module_Info;

    my $return_string;

10    if ($equation =~
    /\^s*\$readmem(b|h)\s*\(\s*\"s*(\S+)\s*\"s*\,\s*(\w+)\s*\)\s*\;\s*$/is)
    {
        my ($mem_radix,$dat_filename,$mem_name) = ($1,$2,$3);
15        return (&ReadMem($mem_radix,
            $dat_filename,
            $mem_name,
            @Module_Info
            )
20        );
    }
    if ($equation =~ /\^s*\$write\s*\(\s*\"(.*)"\s*\,\s*(.*)\s*\)\s*\;/is)
    {
        my ($quote,$values) = ($1,$2);
25        my @values_array = split (/s*\,\s*/s,$values);
        my @string_array = (""); #put null in first spot and index from 1 ..
        $string_length
        my $string_length = 0;
30        while ($quote)
        {
            $string_length++;
            if ($quote =~ s/^(\\%\\)\w//)
            {
35                die "ERROR \$equation ONLY %c IS CURRENTLY SUPPORTED AS DATA
                TYPE!\n"
                unless ($1 =~ /\%c/i);
                push
                (@string_array,"character'val(CONV_INTEGER(".shift(@values_array)."))");
40                }
            else
            {
                $quote =~ s/^(.)//;
                push (@string_array,"'$1'");
45                }
        }
        # make a string signal of width $string_length
        my $string_name = &Get_Exclusive_VHDL_Name("string_name",$Width_List);
        $$Signal_List{$string_name} = "SIGNAL $string_name : STRING(1 TO
50 $string_length);";
        $$Width_List{$string_name} = "1,$string_length";

        #warn "string_array is @string_array\n";
        foreach $index (1..$string_length)
65        {
            $return_string .= "        $string_name($index) <=
            $string_array[$index];\n";
        }
        $return_string .= "        write(output,$string_name);\n";
60        return($return_string);
    }
}

```

```

        die "ERROR Process_Dollar_Verilog_Statements, STATEMENT ($equation) NOT
UNDERSTOOD\n";
    }
}

```

```
5 sub V2VHD_Equation_Wrapper
```

```
{
  my ($equation, @Module_Info) = @_;
```

```

10      my ($Width_List,
        $Signal_List,
        $pWire_Assignments,
        $Equivalence_List) = @Module_Info;

```

```
my %E_List;
```

```
my $pE_List = \%E_List;
$pE_List = $Equivalence_List
    if ($Equivalence_List);
```

```
20 my $string = &V2VHD_Equation(@_, $pE_List);
    $string = &Replace_Equivalences($string, $pE_List);
```

```
#Undef the tmp names so that we can use them again next time
#Do not undef if $Equivalence_List was non-null because
#we've been recursively called.
if (!$Equivalence_List)
```

```
{
    foreach $key (keys(%pE_List))
```

```
{
    #undef ($$Width_List{$key});
    $$Width_List{$key} = "";
}
```

```
#kill std_logic_vector' on lhs concatenations
while ($string =~ s/^\s*std_logic_vector\'//s){;}
```

```
#convert assignment operator to <=
$string =~ s/\={1}\s*/\<= /s;
```

```
40    #warn "    Final Answer: $string\n";
    return ($string);
```

```
#####
}
```

```
45 # Replace
```

```
#
# Replace replaces a complicated expression with a simple exclusive name.
# See V2VHD_Equation comment.
# e.g.
```

```
50 # &Replace ("foo = a + b", 4,@Module_Info);
```

```
# would get an exclusive name for foo, set
# $Equivalence_List{&Get_Exclusive_Name(foo,$Width_List)} = a + b; set
# $Width_List(<the exclusive name for foo>)= 4
```

```

55 # and return the exclusive name for foo so that V2VHD_Equation can substitute
    it
    # into equations.

```

```
sub Replace
```

```
60     my ($equation,  
        $width,  
        @Module_Info) = @_;
```

```

my ($Width_List,
    $Signal_List,
    $pWire_Assignments,
    $Equivalence_List) = @Module_Info;

```

```

5      #die "ERROR Replace: width is ($width) in equation ($equation)\n"
      #unless $width;
      my ($replacement_name,@tmp_replacement_value) = split
        (/s*\={1}\s*/s,$equation);
10     my $replacement_value = join (" = ",@tmp_replacement_value);
        $replacement_name =
        &Get_Exclusive_VHDL_Name("$replacement_name",$Width_List);
        $$Equivalence_List{$replacement_name} = $replacement_value;
        $$Width_List{$replacement_name} = $width;
15     return ($replacement_name);
    }

```

```

#####
#
20 # Find_In_Order
#
# searches $equation for each term in a "|" separated regexp.
# searches in order and returns the first exp that matches.
# returns a \ escaped string so that $equation can be
25 # patterned matched.

sub Find_In_Order
{
30     my ($equation,$regexp) = @_;
        $regexp =~ s/([^\|])$/\1|/s;

        my $tmp_regexp = $regexp;
        my $exp;
        while ($tmp_regexp =~ s/^(.*?([^\|])\|)/)
35         {
            $exp = $1;
            #warn "FIO looking for ($exp)\n";
            if ($equation =~ /($exp)/)
            {
40                 $exp = $1;
                #warn "FIO found ($exp) IN EQUATION ($equation)\n";
                last;
            }
        }
45     die "ERROR Find_In_Order: ($regexp) NOT FOUND IN ($equation)\n"
        unless ($exp ne "");
        $exp =~ s/(\W)/\\$1/g;
        return ($exp);
}

```

```

50 #####
#
# V2VHD_Equation
#
55 # Verilog and VHDL have similar operators, but of course they are
# different because as Buddah says, "Life is Suffering".
#
# Fortunately, since the operators do pretty much the same thing, we
# dont have to evaluate too many verilog expressions. For most
60 # operators, we just need to translate verilog arithmetic into vhd1
# arithmetic.
#
# Verilog and vhd1 handle numbers differently, so we need to convert

```

09880106 "061201

09:00:106.06:1201

```
# numbers from verilog (32,4'hF,0) into their corresponding vhd1
# std_logic_vector bit strings "10000","1111","00". We also need to
# determine the bit width of the number. We do that based upon the
# closest known width operating on the number. i.e. (foo[4:0] == 0)
5 # means 0 probably has width of 5.
#
# VHDL really like "types" and that gets us in a little trouble. A
# naive translation of the verilog expression:
#
#       wire counter_ne_zero = !(counter[3:0] == 0)
10 # Would be:
#
#       SIGNAL counter_ne_zero: STD_LOGIC;
#       counter_ne_zero <= NOT (counter(3 DOWNT0 0) = "0000")
#
# Unfortunately, the result of (counter(3 DOWNT0 0) = 0) is type
15 # boolean, not std_logic. If there were an easy way to type change a
# boolean to std_logic, it would be easy to say:
#       counter_ne_zero <= NOT (To_std_logic (counter(3 DOWNT0 0) =
# "0000"))
#
20 # But I have not found an easy way to do so. To work around this, I
# declare a tmp signal tmp_counter_ne_zero. (I make sure this signal
# isnt already used by someone else first.) It gets assigned to using
# the vhd1 WHEN, ELSE statement:
#       SIGNAL tmp_counter_ne_zero: STD_LOGIC_VECTOR (0 DOWNT0 0) ;
#       tmp_counter_ne_zero <= "1" WHEN (counter(3 DOWNT0 0) = "0000") ELSE
25 # "0";
#       --Now we are back in happy std_logic_vector land:
#       counter_ne_zero <= NOT (tmp_counter_ne_zero);
#
30 # All types are converted to STD_LOGIC_VECTOR inside the Entity (VHDL for
# module)
# We convert types to/from STD_LOGIC at the beginning and end of the entity.
# e.g.
# module foo(c);
35 # output c;
# //more verilog code here
# endmodule
#
# turns into
40 # ENTITY foo IS
# PORT (
#     SIGNAL a : IN STD_LOGIC;
#     SIGNAL b : INOUT STD_LOGIC;
#     SIGNAL c : OUT STD_LOGIC;
45 # );
# END foo;
# ARCHITECTURE behavior OF foo IS
#     SIGNAL tmp_a : STD_LOGIC_VECTOR(0 DOWNT0 0);
#     SIGNAL tmp_c : STD_LOGIC_VECTOR(0 DOWNT0 0);
50 #     --INOUT signals are kept as STD_LOGIC_VECTOR
#     --other signals here
# BEGIN
#     --verilog code translates to vhd1 here
#     c <= tmp_c(0);
55 #     tmp_a(0) <= a;
# END behavior
#
# IMPORTANT IMPORTANT
# INOUT STD_LOGIC_VECTOR (0 DOWNT0 0) signals are not converted to STD_LOGIC
60 # INOUT signals are not so easy to convert to STD_LOGIC
# sometimes the INOUT signal looks like an output
# sometimes its an input. Eventually, I could declare two tmp_signals.
# Convert the signal on the rhs of all equations to be the input version
```

```

# signal and convert all signals on the lhs to be the output signal.
# Determine what signal is driving. Then say:
# inout_signal = (Driving) ? tmp_output_inout: 1'bz;
# tmp_input_inout = inout_signal.
5 # For now you'll just have to deal with a STD_LOGIC(0 DOWNT0 0) INOUT.
# Everything instantiating the module will hook up to it correctly.
# END IMPORTANT IMPORTANT
#
# V2VHD_Equation can recursively and iteratively call itself.
10 # The main idea is that complicated expressions
# can be turned into simpler expressions through replacing
# expressions with exclusive words. (see subroutines Replace and
Replace_Equivalences)
# Converting an expression with V2VHD_Equation simplifies th
15 # e.g.
# verilog: c <= a + b + |c;
# Gets turned into
# 1) c <= a + b + reduction; // $Equivalence_List(reduction) =
"(c(msb) or c (msb - 1) ... or c(lsb))" // $Width_List(reduction) := 1; #as if
20 # reduction were a verilog signal.
# 2) c <= addition + reduction // $Equivalence_List(addition) = "a + b";
# // $Width_List(reduction) =
max(width(a),width(b)) + 1;
25 # 3) c <= tmp_addition // $Equivalence_List(tmp_addition) = addition
+ reduction;
# // $Width_List(reduction) =
max(width(addition),width(reduction)) + 1;
#
30 # V2VHD would then return c <= tmp_addition

# V2VHD_Equation_Wrapper is the top level call to V2VHD_Equation. calls
&Replace_Equivalences after
# V2VHD_Equation. Replace_Equivalences will replace every
35 key(%Equivalence_List) with $Equivalence_List{$key}.
# Using c <= tmp_addition from the example before. Replace_Equivalences will
do:
# 0) c <= tmp_addition
# 1) c <= addition + reduction;
40 # 2) c <= a + b + reduction;
# 3) c <= a + b + (c(msb) or c (msb - 1) ... or c(lsb)); --voila, a valid vhdl
equation

sub V2VHD_Equation
45 {
my ($equation,
@Module_Info) = @_ ;

my ($Width_List,
50 $Signal_List,
$Wire_Assignments,
$Equivalence_List) = @Module_Info;

my $width;
55 my $left;
my $right;

while ($equation =~ s/\s+//sg){};
#Convert all verilog numbers to equivalent word name.
60 while ($equation =~ s/(.*?)(\d+\'[bodh][\da-fxz]+)(.*)/$1/si)
{
my ($replace_this) = &VN2BS($2);
$width = &Width_Of($2,$Width_List);

```

```

$replacement_name      =      &Replace      ("Verilog_Number      =
$replace_this", $width, @Module_Info);

$equation .= " $replacement_name $3";
5   #warn "   verilog number equation now is $equation\n";
}

#Convert vector [max_index:min_index] to equivalent word name.
while ($equation =~ s/^(.*?\b)(\w+)\s*([[(.*?)\]) (.*)/$1/s)
10  {
    my $replace_this = $2;;
    ($replace_this, $left, $right) = &Vector_Range("$2$3", $Width_List);
    my $rest = $5;
    #Worry about memories
15   my ($l, $r, $up, $down) = split (/s*\.,s*/s, $$Width_List{$2});
    if ($up ne "")
    {
        #It's a memory
        #warn "equation is $equation\n";
20   $2\n";
        warn "\nVERILOG TO VHDL CONVERSION: ".&date_time." USING MEMORY
        $replace_this .= "(CONV_INTEGER(unsigned($4)))";
        ($left, $right) = ($l, $r);
    }
    else
    {
        $right = $left if ($right eq ""); #If only one value,
        #$replace_this .= &Vector_Order$left, $right);
        $replace_this .= "[$left:$right]"; #we'll change vector order when we
30   #replace_equivalences
    }
    #warn "in brackets, (@Module_Info)\n";
    $replacement_name      =      &Replace("Verilog_Bracket      =
$replace_this", "$left, $right", @Module_Info);
35   $equation .= "$replacement_name $rest";
}

#WRONG WRONG WRONG WRONG WRONG
#THE ORDER BELOW IS WRONG
40   # Now the order of operators we have (according to verilog 1364 and Samir
Palnitkar's book)
# to handle are
# 1) Parentheses, Replecation and Concatenation
# 2) Unary (+, -, !, ~)
45   # 3) Multiply, Divide, Modulus (*, /, %)
# 4) Add, Subtract, Shift (+, -), <<, >>
# 5) Relational (<, <=, >, >=)
# 6) Equality (==, !=)
# 7) Binary Bitwise (&, ^, |)
50   # 8) Logical (&&, ||)
# 7) Reduction (&, ^, |)
# 9) Conditional (?:)
#10) Assignment (=)

55   #WRONG WRONG WRONG WRONG WRONG
#THE ABOVE ORDER IS WRONG
#AND IS A BUG IN THE 1364 spec.

60   #SEE THE FOLLOWING POST FROM ALT.COMP.LANG.VERILOG

#...Based on the information obtained from "private sources" (a member
#of the 1364 WG) I concluded that the Table 4-4 is, indeed,
#wrong, and, maybe, the table in Thomas and Moorby 2nd is not

```

00330106 "061201

```

#right, either.
#ALL unary operators have precedence higher than ANY binary operator,
#while the precedence among unary operators is non-essential.
#So, Table 4-4 gives erroneous precedence for ~& and ~| UNARY
#operators placing them where such BINARY operators should have
#been had they existed. It is still to be determined whether
#binary ^~/^ has the same precedence as & (T&M) or lower (IEEE
#Draft), and that can be easily accomplished with Verilog-XL
#which I'm going to do shortly.
#
#Regards to all,
#Sergei Sokolov

```

```

15 my $unary_operators = '\+|\-|!|\~';
    my $reduction_operators =
'\&{1}|\~\&{1}|\^ {1}|\~\^ {1}|\^ \~ {1}|\| {1}|\~ \| {1}';
    my $all_unary_operators = join ('', $unary_operators, $reduction_operators);
    my $relational_equality_operators = '\<|=|\<|\>|=|\>|\={2}|\!|=|';
20 my $arithmetic_operators = '\*|\/|\+|\-';
    my $binary_bitwise = '\&{1}|\^ {1}|\| {1}|\~\^|\^ \~';
    my $shift_operators = '\<\<|\>\>';

    #It's a pain to differentiate between \&& and \&, so change
    #logical operators to something easier to discern. The ` marks
    #are the verilog escape character, so we know some verilog typer
    #won't accidentally type `AND` accidentally.
    while ($equation =~ s/\&{2}/`AND`/){};
    # Similar argument for ||
30 while ($equation =~ s/\| {2}/`OR`/){};
    my $logical_operators = '`AND`|`OR`';

    #I also hate those === and !== operators which (for all synthesizable
    #code does just the same thing as == and !=
35 while ($equation =~ s/\!|= {2}/\!|=/){};
    while ($equation =~ s/\= {3}/\= =/){};

    my $binary_operators_with_same_width_operands_and_shift_operators = join
40 ('|',
$arithmatic_operators,
$binary_bitwise,
$shift_operators,
45 $relational_equality_operators,
$logical_operators
);

50 my $operator_order = join ('|', $all_unary_operators,
$binary_operators_with_same_width_operands_and_shift_operators,
$logical_operators);

55 # 1a) PARENTHESES
    (
    my ($before_paren, $inside_paren, $after_paren) =
    &Count_Parentheses($equation);
    while ($inside_paren ne "")
60 {
    #####
    # if there is one or more operators inside the parentheses, then
    process it(them).

```

09880106-061201
102790-9070886

09080106-061201

```
# Otherwise strip the parentheses and continue.
# i.e. (counter - 1), process it.
# (counter) , return counter
if ($inside_paren =~ /$operator_order/)
5 {
    #warn "replace_this found ($inside_paren) in ($equation)\n";
    #Recurse equation (DAMN DAMN equation!);
    my $replace_this = &V2VHD_Equation($inside_paren, @Module_Info);
    #warn "Parentheses converted ($inside_paren) to ($replace_this),
10 ($$Width_List{$replace_this})\n";
    die "ERROR V2VHD_Equation: replace_this returned ($replace_this)\n"
        unless ($replace_this =~ s/^\s*(\w+)\s*$/\1/s);
    my $replacement_name = &Replace("Parentheses = $replace_this",
        $$Width_List{$replace_this},
15 @Module_Info);
    $equation = "$before_paren $replacement_name $after_paren";
    #warn "equation now is $equation\n";
}
else
20 {
    #warn " parentheses, no operator found\n";
    #put parentheses around
    $equation = "$before_paren $inside_paren $after_paren";
}
25 ($before_paren,$inside_paren,$after_paren) =
    &Count_Parentheses($equation);
    #warn " parentheses equation now is $equation, width is $width\n";
}
}
30 # 1b) REPLECATION AND CONCATENATION
{
    #Replicate first
    #warn " replecation equation was $equation\n";
    while ($equation =~ s/^(.*)\{\s*(\d+)\s*(\{.*\})/\1/s)
35 {
        my $before_curly_brace = $1;
        my $repeat_number = $2;
        my ($b,$m,$e) = &Count_Parentheses($3,'{\s*','\}\s*');
        #warn "bme is ($b),($m),($e)\n";
        $m = ("m\," x $repeat_number);
        $m =~ s/\\,\s*$/\s/; #lose last comma.
        $e =~ s/^\s*\}\s*$/\s/;
        $equation = "$before_curly_brace $b{$m}$e";
45 #warn " replecation equation now is $equation\n";
    }

    #All other curly_braces ({,}) (including the replecation we handled
    above) are concatenation
50 my ($before_curly_brace,$inside_curly_brace,$after_curly_brace) =
    &Count_Parentheses($equation,'{\s*','\}\s*');
    while ($inside_curly_brace ne "")
    {
        #warn "BPC ($inside_curly_brace) \n";
55 #warn "%$Width_List;
        my $replacement_name = &Process_Concatenation
            ("\"{$inside_curly_brace}\"",@Module_Info);
        $equation = "$before_curly_brace $replacement_name
            $after_curly_brace";
60 ($before_curly_brace,$inside_curly_brace,$after_curly_brace) =
        &Count_Parentheses($equation,'{\s*','\}\s*');
    }
}
```



```

# 2) UNARY OPERATORS
(
#####
5   # In addition to the normal unary operators
   # Verilog has these weird but useful unary operators called reduction
operators
   # a = |foo[3:0] => a = foo[3] | foo[2] | foo[1] | foo[0];
   # It turns out that these operators are quite useful.
10  # However it is difficult to distinguish between the unary reduction
operator
   # and the binary "or" operator.  a = c | foo or a = c + |foo looks quite
like a = |foo.
   #
15  # Here is how we differentiate.  If the first non spaced character before
the possible
   # reduction character is a word (\w) then consider it a binary operator.
If it is a
   # non word character, consider it a unary operator.  This is good even
20  for equations with
   # parentheses because we've already converted parentheses to words.
   # Consider:  a = (c + b) | foo
   # Above, we've already converted (c + b) to tmp_parentheses so the
equation we see is
25  #           a = tmp_parentheses | foo //binary or
   #

   while ($equation =~ /($all_unary_operators)/)
   (
30     my $operator = &Find_In_Order($equation,$all_unary_operators);
                           last unless ($equation =~
s/^(.*?[^\\w\\s])\\s*($all_unary_operators)\\s*(\\w+)\\s*(.*)/$1/s ||
                           $equation =~
s/^(\\s*)($all_unary_operators)\\s*(\\w+)\\s*(.*)/$1/s);
35     my $beginning = $1;
     $operator = $2;
     my $name = $3;
     $width = &Width_Of($name,$Width_List);
     my $rest = $4;
40     my $replace_this;
     if ($operator =~ /^$unary_operators$/)
     {
       if ($operator eq "\\!")
       {
45         if ($width == 1){$replace_this = " (NOT $name)";}
         else{$replace_this = "\\($name \\= \\\".\"(\"0\" x $width).\"\\\"\\\"";}
         $width = 1;
       }
       else
50       {
         $operator =~ s/~/NOT/;
         $replace_this = "($operator $name)";
       }
     }
55     else
     {
       my $value_to_reduce = $name;
       my $reduction_operator = $operator;

60       $reduction_operator = "OR" if ($reduction_operator =~ /^\\|{1}$/);
       $reduction_operator = "AND" if ($reduction_operator =~ /^\\&{1}$/);
       $reduction_operator = "NOR" if ($reduction_operator =~
/^\\~{1}\\|{1}$/);

```



```

                                my          $operator          =
&Find_In_Order($equation,$binary_operators_with_same_width_operands_and_shift_o
perators);
5      die "BINARY_OPERATOR ($operator) not found in ($equation)\n"
      unless ($equation =~ s/^(.*?)(\w+)\s*($operator)\s*(\w+)(.*)/$1/s);
      my $left_operand = $2;
      $operator = $3;
      #warn "operator now is $operator\n";
      my $right_operand = $4;
10     my $rest = $5;
      my $replace_this;
      if (&Is_real($left_operand) && &Is_real($right_operand))
      {
15         #warn " relational equation is real\n";
         #If both are real numbers, evaluate using perl's
         #binary operators.
         if ($operator eq "\`AND\`")
         {
20             if (($2 == 0) || ($4 == 0)){$equation .= "0";}
             else{$equation .= "1";}
         }
         else
         {
25             if ($operator eq "\`OR\`")
             {
                 if (($2 != 0) || ($4 != 1)){$equation .= "1";}
                 else {$equation .= "0";}
             }
             else
30             {
                 my $evaluatedExpression = eval ("$left_operand $operator
$right_operand");

                 # Programming Perl p. 87:
35                 # 'The equal and not-equal operators return 1 for true, and
                 "" for false
                 # just as the relational operators do).'
                 # Translate all forms of "false" to 0:
                 $evaluatedExpression = "0" if (!$evaluatedExpression);
40                 $equation .= $evaluatedExpression;
             }
         }
         $equation .= $rest;
45     next;
}

#shift_operators operate on integers, all other operators need their
integers to be converted to bit_vectors.
50     if ($operator =~ /^$shift_operators$/)
    {
        if (&Is_real($left_operand))
        {
            $replacement_name = eval "$left_operand $operator
55 $right_operand";
            # Translate all forms of "false" to 0:
            $replacement_name = "0" if (!$replacement_name);
        }
        else
60     {
                                my          ($name,$left,$right)          =
&Vector_Range($left_operand,$Width_List);
                                $width = &Width_Of($left_operand,$Width_List);

```

09880106 "061201

09280106-061201
TOTAL 90 902801

```

die "ERROR V2VHD_EQUATION ($equation) HAS UNKNOWN width. UNABLE
TO SHIFT\n"
    if ($left eq "");
5
    die "ERROR V2VHD_EQUATION ($equation) HAS NON-INTEGERSHIFT
AMOUNT\n"
    unless ($right_operand =~ s/^\s*(\d+)\s*$/\s1/s);

10
    if ($right eq "")
    {
        $name = &Add_Intermediate_Signal ("shift = $left_operand",
                                           $left_operand_left_index,
                                           @Module_Info
                                           );
15
        $left--;
        $right = 0;
    }

    #####
20
    #c [3:0] >> 1 = c [3:1];
    #c [0:3] >> 1 = c [0:2];

    my @array = &Order($left,$right);
    my $zeros;
    while ( ($right_operand--) && ($width != 0) )
    {
        if ($operator eq "<<"){$zeros .= "0"; $width++;}
        else{
30
            if ($operator eq ">>"){$pop(@array);$width--}
            else {die "ERROR V2VHD_EQUATION, UNKNOWN SHIFT
ASSIGNMENT($operator)\n";}
        }
    }

35
    my $expand_this_string;
    if (@array)
    {
        $left = $array[0];
        $right = $array[-1]; #last value in the array
        $expand_this_string = "$name\[$left\:$right\]";
40
        if ($zeros ne "")
        {
            $expand_this_string .= ", \"$zeros\"";
        }
45
        #warn "sending $expand_this_string to EAOBVISB\n";
        ($replace_this,$width)
        &Expand_Array_Of_Bit_Vectors_Into_Separated_Bits ("", "",
50
        $expand_this_string,
        @Module_Info

        $replace_this = "std_logic_vector' ( $replace_this)\n";
        );
55
    }
    else
    {
        $replacement_name = "0";
    }
60
}
my $replacement_name = &Replace("shift
$replace_this",$width,@Module_Info);

```

```

    $equation .= "$replacement_name $rest";
    next;
}

5      if ($operator =~ /^$logical_operators$/)
    {
        #operator transforms
        #warn "found lo ($left_operand,$operator,$right_operand)\n";
        $operator = "AND" if ($operator eq "\`AND\`");
10     $operator = "OR" if ($operator eq "\`OR\`");
        $replace_this =
        "(. &Process_If_Condition($left_operand,@Module_Info).) $operator (
            &Process_If_Condition($right_operand,@Module_Info).)";
        $width = 1;
15     my $replacement_name = &Add_Intermediate_Signal ("logical = \"1\"
WHEN ($replace_this) ELSE \"0\"",
                                $width,
                                @Module_Info);

20     $equation .= "$replacement_name $rest";
    next;
}

25     #warn "calling CV2SLV ($left_operand,$operator,$right_operand)\n";
        ($width,$left_operand,$right_operand) =
        &Convert_Integers_To_Std_Logic_Vector($left_operand,
                                                $operator,
30     $right_operand,
        $width_List,
        $Equivalence_List);
35     if ($operator =~ /^$arithmetic_operators$/)
    {
        # VHDL thinks c = a + b results in c_width = max(a_width,b_width).
        # But that is clearly wrong. In reality, c_width =
        max(a_width,b_width) + 1.
40     # So we use reconcile widths to give us signals (new_a,new_b) that
        have widths (a_width+1,b_width+1);
        # We use reconcile known widths to get us a signal of the correct
        width.
        my $new_width;
45     if ($operator =~ /(\+|\-)/)
    {
        $new_width = $width + 1; #Max_width from
        Convert_Integers_To_Std_Logic_Vector above.
        $left_operand = &Reconcile_Known_Widths(
50     $left_operand,
        $left_operand,
        $new_width,

        &Width_Of($left_operand,$width_List),
55     @Module_Info
        );

        $right_operand = &Reconcile_Known_Widths(
60     $right_operand,
        $right_operand,
        $new_width,

        &Width_Of($right_operand,$width_List),

```

09280106-061201

```
@Module_Info
);
```

```
}
```

```
5      if ($operator =~ /\*/)
      {
          $new_width = &Width_Of($left_operand,$Width_List) +
&Width_Of($right_operand,$Width_List) ;
      }
10     if ($operator =~ /\//)
      {
          $new_width = &Width_Of($left_operand,$Width_List) -
&Width_Of($right_operand,$Width_List);
      }
15     #warn "worop is ".$Width_Of($right_operand,$Width_List).", nw is
$new_width\n";

    $width = $new_width;
    #warn "width is ($width)\n";
20    $replace_this = "$new_left_operand $operator $new_right_operand";
    $replace_this = "$left_operand $operator $right_operand";
    my $replacement_name = &Replace("arithmetic" =
$replace_this",$width,@Module_Info);
    $equation .= "$replacement_name $rest";
    next;
25 }

    #relational operator transforms
    if ($operator =~ /^$relational_equality_operators$/)
30    {
        #####
        # Convert boolean (in)equality. Since the output of a conditional
is a boolean
        # and we only understands std_logic, we need to replace the
35 conditional with
        # a wire named something like tmp_std_logic.
        # Then we set tmp_std_logic <= '1' when (condition) else '0';
        #
        # N.B. In the future, since we convert everything to
40 std_logic_vector, we might
        # be able to convert the boolean to std_logic_vector

        my $tmp_left =
&Replace_Equivalences($left_operand,$Equivalence_List);
45     my $tmp_right =
&Replace_Equivalences($right_operand,$Equivalence_List);

        my $tmp_output_name;
        #warn "operator ($operator) is relational_operator\n";
50     if ($operator =~ s/\s*\(=\{2\}\s*/ \s*= \s*/s) {$tmp_output_name =
"$tmp_left\_eq\_tmp_right";}
        if ($operator =~ s|\s*\!\s*=\{1,2\}\s*| \s*\!= \s*/s) {$tmp_output_name =
"$tmp_left\_ne\_tmp_right";}
        if ($operator =~ /\s*<\s*/s) {$tmp_output_name =
55 "$tmp_left\_lt\_tmp_right";}
        if ($operator =~ /\s*<=\s*/s) {$tmp_output_name =
"$tmp_left\_le\_tmp_right";}
        if ($operator =~ /\s*>\s*/s) {$tmp_output_name =
"$tmp_left\_gt\_tmp_right";}
60     if ($operator =~ /\s*>=\s*/s) {$tmp_output_name =
"$tmp_left\_ge\_tmp_right";}

        if ($tmp_output_name)
```

09880105-061201

```

    {
        my $boolean = "$left_operand $operator $right_operand";
        $tmp_output_name = &Add_Intermediate_Signal("$tmp_output_name =
5  \"1\" WHEN \"($boolean)\" ELSE \"0\"",
                                1,
                                @Module_Info);

        $equation .= "$tmp_output_name$rest";
        #warn "ton, ($equation)\n";
10    }
    next;
}

$operator = "XOR" if ($operator =~ /\^\^$/);
15 $operator = "AND" if ($operator =~ /\^\&$$/);
$operator = "OR" if ($operator =~ /\^\|$/);
$operator = "XNOR" if ($operator =~ /\^\^~$/);
$operator = "XNOR" if ($operator =~ /\^\~\^$/);

20 my $replacement_name = &Replace("binary_operator = $left_operand
$operator $right_operand",
                                $width,
                                @Module_Info);
    $equation .= "$replacement_name$rest";
25 }
}

#CONDITIONAL ?:
while ($equation =~ s/((\s*)(\w+)\s*\?\s*(\w+)\s*\:((\s*)/"$1$3    WHEN
30 \(".&Process_If_Condition($2,@Module_Info)."\) ELSE $4"/sge)
{
    #warn "found conditional($2)\n"
    ;}

# ASSIGNMENT
35 # The following is copied very closely from the
$binary_operators_with_same_width_operands case above
# Assignment gets a little bit nasty because unlike verilog, VHDL requires
that the left and right sides of
# assignments be the same width. So we need to chop off or add bits to the
40 right hand side of the assignment.
# We do that in &Reconcile_Widths.
{
    my $tmp_equation = $equation;
    if ($equation =~ s/^(.*) (\w+) \s* (\={1}) \s* (\w+) (.*) /$1$2 $3/s)
45 {
        my $left_operand = $2;
        my $operator = $3;
        my $right_operand = $4;
        my $rest = $5;

50 ($width,$left_operand,$right_operand) =
&Convert_Integers_To_Std_Logic_Vector($left_operand,
                                $operator,
                                $right_operand,
55 $width_List,
$Equivalence_List);

60 #warn "    left_operand, operator, right_operand, rest, width,
replace_this IS $left_operand, $operator, $right_operand, $rest, $width,
$replace_this\n";
    $right_operand = &Reconcile_Widths($left_operand,

```

09880106 "061201

09880106-061201

```

                                $right_operand,
                                @Module_Info
                                );
5      #warn " ro is $right_operand\n";
      #operator transforms
      #replace_this = "$operator $right_operand";
      $equation .= "$right_operand$rest";#replace_this$rest";
      #warn " assignment equation now is $equation\n";
    }

10     #We also need to match up lhs with values following ELSE statements
     #I put a `` mark before all processed ELSE words to keep us from an
infinite loop.
     #Again, we need to Reconcile_Widths.
15     while ($equation =~
s/^(.??)(\w+)(\s*\={1}.*?ELSE(\s+|[\^\\`w]))(\w+)(.*)/$1/s)
    {
20       my $left_operand = $2;
       my $in_between = $3;
       my $right_operand = $5;
       my $rest = $6;

                                   ($width,$left_operand,$right_operand) =
&Convert_Integers_To_Std_Logic_Vector($left_operand,
                                   "=",
25     $right_operand,
     $Width_List,
30     $Equivalence_List);
       #warn " Conditional Assignment: left_operand, in_between,
right_operand, rest, width, replace_this IS $left_operand, $in_between,
$right_operand, $rest, $width, $replace_this\n";

35       #Do the same width reconciliation deal as above. (if needed)
       $right_operand = &Reconcile_Widths($left_operand,
                                   $right_operand,
                                   @Module_Info
                                   );

40       #in_between transforms
       $replace_this = "$left_operand $in_between \`\`\`$right_operand";
       $equation .= "$replace_this$rest";
       #warn " ELSE assignment equation now is $equation\n";
45     }

     #crush `marks
     while ($equation =~ s/\`//g){}

50     $equation =~ s/^\s*(.??)\s*$/$1/s;

     return($equation);
  }

55  #####
  #
  # Reconcile_Widths and Reconcile_Known_Widths
  #
60  # A big difference between verilog and vhd1 is that verilog is more lenient
when it
# comes to width assignments.
# eg. foo[7:0] = ook[3:0] => foo[7:4] = {0,0,0}; foo[3:0] = ook[3:0]
```


09880105"061201
T02190"

```
# eg. foo[3:0] = ook[7:0] => foo[3:0] = ook[3:0];
#
# Not so for vhdl. It requires left_side_width = right_side_width. Thus this
function.
5 # Reconcile_Widths, takes as input a left and right operands. It forces the
# right operand to the same width as the left operand.
#
# If the left operand width is smaller than the right operand width
# Reconcile_Widths creates a signal_of_right_side_width <= right_side;
10 # then sets left_side <= (signal_of_right_side_width)((left_side_width -1)
DOWNTO 0).
#
# If the left operand width is bigger than the right operand width,
# Reconcile_Widths creates a signal_of_right_side_width <= right_side,
15 # It sets left_side <= std_logic('0'x($left_side - $right_side),
signal_of_right_side_width);
#
# eg. foo[7:0] = ook[3:0] becomes foo(7 DOWNTO 0) <= &V2VHD_Equation (foo[7:0]
= {4'd0,ook[3:0]})
20 #
# Reconcile_Widths wraps around Reconcile_Known_Widths
# Reconcile_Known_Widths can also be used when one is comparing the right_side
to a left_side
# that is not in %$Width_List. e.g. when instantiating a Module.
25 #####
#

sub Reconcile_Widths
(
30 my ($left_operand,
    $right_operand,
    @Module_Info) = @_;

    my ($Width_List,
35 $Signal_List,
    $pWire_Assignments,
    $Equivalence_List) = @Module_Info;

    my ($left_width) = &Width_Of($left_operand,$Width_List);
40 my ($right_width) = &Width_Of($right_operand,$Width_List);

    die "ERROR Reconcile_Widths: WIDTH OF ($left_operand (" .
&Replace_Equivalences($left_operand, $Equivalence_List) . ")) NOT KNOWN!\n" if
($left_width eq "");
45 die "ERROR Reconcile_Widths: WIDTH OF ($right_operand (" .
&Replace_Equivalences($right_operand, $Equivalence_List) . ")) NOT KNOWN!\n" if
($right_width eq "");

    return (&Reconcile_Known_Widths ($left_operand,
50 $right_operand,
    $left_width,
    $right_width,
    @Module_Info
    )

55 );
}

sub Reconcile_Known_Widths
(
60 my ($left_operand,
    $right_operand,
    $left_width,
    $right_width,
```

```

@Module_Info) = @_;

my ($Width_List,
    $Signal_List,
5    $pWire_Assignments,
    $Equivalence_List) = @Module_Info;

#warn ("RNW, $left_operand,
10    #$right_operand,
    #$left_width,
    #$right_width\n");

#We may not know the left or right operand on pass 1,
#die "Reconcile_Widths, left_width not known" if ($left_width eq "");
15 #die "Reconcile_Widths, right_width not known" if ($right_width eq "");
return ($right_operand) if ($left_width eq "");
return ($right_operand) if ($right_width eq "");

my $tmp_signal;
20 if ($left_width == $right_width)
{
    #warn "$left_operand has same width as $right_operand, returning\n";
    return ($right_operand);
}
else
25 {
    $tmp_signal = &Add_Intermediate_Signal
("$left_operand\_ $right_width\_bits\_wide = $right_operand",
    $right_width,
30    @Module_Info);
}

if ( $left_width < $right_width)
{
35    #warn " RW: lw < rw.\n";
    $right_operand = "$tmp_signal (($left_width - 1) DOWNT0 0)";
}

if ( $left_width > $right_width)
40 {
    my $width_difference = $left_width - $right_width;
    my $filler = "\0\0\0\0" x $width_difference;
    my ($EAOBV,$junk) =
&Expand_Array_Of_Bit_Vectors_Into_Separated_Bits("",$tmp_signal,@Module_Info);
45    $right_operand = "std_logic_vector\' ($filler$EAOBV)";
}
return ($right_operand);
}

50 #####
#
# Add_Intermediate_Signal takes a vhdl string signal "a" or "a = b" and
# returns
# the correct signal name to use. If signal is of the form "a = b" it replaces
55 # equivalences of b and adds a <= b to wire_assignments

sub Add_Intermediate_Signal
{
60    my ($signal,
        $width,
        @Module_Info) = @_;

    my (

```

0980106-061201

09880106-061201

```

    $Width_List,
    $Signal_List,
    $pWire_Assignments,
    $Equivalence_List) = @Module_Info;
5
    my ($signal_lhs, @signal_rhs) = split (/^s*\(=1\)s*/s,$signal);
    my $signal_rhs = join ("=",@signal_rhs);
    $signal_rhs = &Replace_Equivalences($signal_rhs,$Equivalence_List)
        if ($Equivalence_List && $signal_rhs);
10
    die "ERROR Add_Intermediate_Signal: ILLEGAL WIDTH ($width) IN SIGNAL
    ASSIGNMENT ($signal)\n"
        unless (($width =~ s/^s*(\d+)\s*/$1/) && ($width > 0));

15
    #just return signal_rhs if only one value is on the rhs.
    #eg. foo = bar, just return bar
    if ($signal_rhs =~ s/^s*(\w+)\s*/$1/s)
    {
        #Orion, do width checking here
20
        return ($signal_rhs);
    }

    #rename signal_lhs
    $signal_lhs = &Get_Exclusive_VHDL_Name($signal_lhs,$Width_List);
    if ($signal_rhs ne "")
    {
        $pWire_Assignments .= " $signal_lhs <= $signal_rhs;\n";
    }

30
    $$Signal_List{$signal_lhs} = "SIGNAL $signal_lhs :
    STD_LOGIC_VECTOR(".$width-1)." DOWNTO 0);";
    $$Width_List{$signal_lhs} = ($width-1).",0";
    return ($signal_lhs);
}

35
#####
#
# Is_real returns 1 if the value passed to it is a real number, i.e integer >=
0
40
#####
#
sub Is_real
{
    my $value = shift (@_);
45
    return (1) if ($value =~ /^s*\d+s*/s);
    return (0);
}

50
#####
#
# Count_Parentheses
#
# so i have a string always @(blow(me)(leonardo|synplicity)) blerg (boof)
# I want to perform computations on the string surrounded by the
55
# beginning and last parentheses. I call Count_Parentheses and it
# returns 3 values, the beginning string: "always @", the parenthesized string
# "blow(me)(leonardo|synplicity)" and the last string "blerg (boof)".
#
# If I want to search on something other than parentheses, say begin,end, I can
60
# place their values in $begin_match and $end_match.
#####
#
sub Count_Parentheses
```

```

{
    my ($string,$begin_match,$end_match) = @_;
    my $begin_string;
    my $paren_string;
5    my $end_string;
    my $begin_match_default = '\s*\(\s*';
    my $end_match_default   = '\s*\)\s*';
    $begin_match = $begin_match_default unless ($begin_match);
    $end_match   = $end_match_default unless ($end_match);
10
    warn "                CP string is $string,\n"
        ".      $begin_match,\n      $end_match\n"
        if (0);
    #if ($begin_match ne $begin_match_default);
15
    return("", "", "$string")
        unless ($string =~ /^(*?)$begin_match(.*)$/s);

    #warn "                                found first bit $1, \n
20    rest is $2\n      in string $string\n"
    #if ($GLOBAL_WARNING);
    # ".      $begin_match,\n      $end_match\n"
    #if ($begin_match ne $begin_match_default);

25    $begin_string = $1;

    my $paren_count = 1;
    $end_string = $2;

30    while ($end_string =~ s/^(.*?)($begin_match|$end_match)(.*)$/3/s)
    {
        my $match;
        $match = $2;
        $paren_string .= $1;
35
        if ($match =~ /$begin_match/)
        {
            $paren_count++;
        }
        else
        {
40            $paren_count = $paren_count - 1;
        }

45        last if ($paren_count == 0);
        #else
        $paren_string .= $match;
    }

50    #die "mismatched      $begin_match,$end_match      in      string
    $begin_string$paren_string$end_string" if ($paren_count != 0);
    return ($begin_string,$paren_string,$end_string);
}

55 sub Handle_Next_If_Else_Line
{
    my ($if_body, @Module_Info) = (@_);

60    my (
        $Width_List,
        $Signal_List,
        $pWire_Assignments,
        $Equivalence_List) = @Module_Info;

```

09880106 0512001

```
my $rest_of_module;

my $spacing;
#####
5  # following a verilog "if(condition)", "always @(condition), else,
statement, there are three things that
# can follow the statements.
# 1) an if statement
10 # 2) a begin - end block
# 3) a one line statement ending with a semi-colon;

# 1) an if statement
if ($if_body =~ /^(\\s*)\\bif\\s*(\\(.*\\))/s)
15 {
    my ($tmp,$if_conditions,$tmp_if_body) = &Count_Parentheses($2);
    #warn "if statement is ($if_conditions)\\n";
    $if_body = "$1IF ".&Process_If_Condition($if_conditions,@Module_Info).
THEN ";
20 ($tmp_if_body,$rest_of_module)
&Handle_Next_If_Else_Line($tmp_if_body,@Module_Info);
    $if_body .= $tmp_if_body;

    #If an else statement follows the block
    #Process it.
    if ($rest_of_module =~ /^(\\s*)else(.*\\s)/s)
    {
        $if_body .= "$1ELSE";
        ($tmp_if_body,$rest_of_module)
30 &Handle_Next_If_Else_Line($2,@Module_Info);
        $if_body .= $tmp_if_body;
    }
    $if_body .= "\\n$spacing"."END IF;";
    return ($if_body,$rest_of_module);
35 }

# 2) a begin - end block
if ($if_body =~ /^(\\s*)\\bbegin\\b/s)
40 {
    #warn "hniel, begin end line, ib is $if_body\\n"
    #if ($HNIEL_DEBUG);
    my ($tmp,$tmp_body,$rest_of_module)
    &Count_Parentheses($if_body,'\\bbegin\\b','\\bend\\b');
45 #if we are in a begin_end block, then process each command. If there is
an if statement
    #lurking, then call this function again.

    #my @commands = split (/\\;/s,$tmp_if_body);
    my $line;
    $if_body = "";
    while (!(($tmp_body =~ /^(\\s*)$/s))
50 {
        ($tmp,$tmp_body) = &Handle_Next_If_Else_Line($tmp_body,
55 @Module_Info);
        $if_body .= $tmp;
        #warn "begin_end, ib($if_body)\\n--tmp_body($tmp_body)\\n--\\n"
        #if ($HNIEL_DEBUG);
    }
    #warn "hniel, ib now is ($if_body)\\n"
    #if ($HNIEL_DEBUG);
    return ($if_body,$rest_of_module);
60 }
}
```

```

# 3) a one line statement ending with a semi-colon;
if ($if_body =~ /^(\\s*)(.*?;)(.*)/s)
{
5   #warn "hniel, oneline ($2)\\n"
    #if ($HNIEL_DEBUG);
    $if_body = $1.&Process_Register_Assignment($2,@Module_Info);
    #warn "          becomes ($if_body)\\n"
    #if ($HNIEL_DEBUG);

10   $rest_of_module = $3;
    return ($if_body,$rest_of_module);
}
}

15 #####
#
# Process_If_Condition: converts verilog if conditions
# to VHDL IF conditions.
20 # V2VHD_Equation
#####
#
sub Process_If_Condition
{
25   my ($condition, @Module_Info) = (@_);
    my (
        $Width_List,
        $Signal_List,
        $pWire_Assignments,
        $Equivalence_List) = @Module_Info;

30   #warn "PIC, $condition\\n";
    my %E_List;
    if ($Equivalence_List eq "")
    {
35       $Equivalence_List = \\%E_List;
        $Module_Info[3] = $Equivalence_List;
    }
    #relational_equality_operators, and transforms copied from V2VHD_Equation;
40   my $vhdl_relational_equality_operators = '\\<\\=|\\<|\\>\\=|\\>|\\={1}|\\|\\=';
    # If there is no relational_equality_operators in the equation, it's
    # implied that the equation != 0. i.e.
    # if (foo) //same thing as if (foo != 0);
    # if (!foo) // same thing as if (!foo != 0);
45   my $result = &V2VHD_Equation($condition,@Module_Info);

    my $result_width = &Width_Of($result,$Width_List);
    #die "ERROR Process_If_Condition, WIDTH FOR ($result) NOT KNOWN in
50 ($condition)!\\n"
    #unless ($result_width);
    my $rhs = 0 x $result_width;

    $result = &Replace_Equivalences($result,$Equivalence_List);
    my $return_string = "$result";
55   $return_string = "($result) \\|\\= \\$rhs\\"
        unless ($result =~ /$vhdl_relational_equality_operators/);
    #warn "PIC, returns $return_string\\n" if $start_special;
    return ($return_string);
}

60 #####
#
# Get_Attribute_Types: Returns all attributes defined by VHDL code.

```

T02190"061201

09880106-061201

```
# Currently, I have just pasted in the attribute definitions from exemplar.vhd
# into a "HERE" string;
#####
#
5  sub Get_Attribute_Types
  (
    my ($pAttribute_List) = @_ ;

    my $vhdl_string = q[
10      ---
      -- Attribute declarations
      ---

      attribute required_time    : time ;
15      attribute arrival_time    : time ;
      attribute output_load      : real ;
      attribute max_load         : real ;
      attribute clock_cycle      : time ;
      attribute clock_offset     : time ;
20      attribute pulse_width     : time ;
      attribute input_drive      : time ;
      attribute nobuff           : boolean ;
      attribute pin_number       : string ;
      attribute preserve_signal  : boolean ;
25      attribute noopt           : boolean ;

      -- New attributes in 2.1 release

      -- Specify pin_numbers for bits of a 1-dimensional array
30      type exemplar_string_array is array (natural range <>,
natural range <>) of character ;
      attribute array_pin_number : exemplar_string_array ;

      -- Buffer_sig attribute to force a (clock) buffer on a signal
35      attribute buffer_sig : string ;

      -- PAD attribute to force a particular PAD cell on a IO pin
      -- Does not work for Xilinx, Orca and Altera.
40      attribute pad : string ;

      -- Type needed to indicate speed requirements for module
generators

      type modgen_select is (smallest, small, fast, fastest);
45      -- Use this attribute to set speed on signals/variables
      attribute modgen_sel : modgen_select ;

      -- New attributes in 2.2 release

      -- Attributes for encoding of enumerated types.
50      type encoding_style is (BINARY, ONEHOT, TWOHOT, GRAY, RANDOM)
;
      attribute TYPE_ENCODING_STYLE : encoding_style ;

55      -- Example of using type_encoding_style for an enumerated
type :
      --      type state_t is (PLAY, WAIT_FOR_MOVE, END_OF_GAME);
      --      attribute TYPE_ENCODING_STYLE of state_t: type is

60      ONEHOT ;

      attribute TYPE_ENCODING : exemplar_string_array ;
```

```

-- Example of using TYPE_ENCODING for an enumerated type :
--     type state_t is (PLAY, WAIT_FOR_MOVE, END_OF_GAME);
--         attribute TYPE_ENCODING of state_t:type is
5 ("011","110","101") ;
    ];

```

```

#Crush comments
while ($vhdl_string =~ s/^(.*?)\-\-\-.*$/\1/m){;}

```

```

10 my @commands = split (/\\s*\\;\\s*/s,$vhdl_string);
    foreach $command (@commands)
    {
        if ($command =~ /^\\s*attribute\\s+(\\w+)\\s+:\\s*(\\w+)/s)
        {
15             $$pAttribute_List{$1} = $2;
            #warn "Type $1 is $2\\n";
        }
    }
}

```

```

20 #####
#
# V2VHD converts from a synthesizable verilog file to a synthesizable vhd1 file
#
# Still to be done
# 1) Make multiple behaviour modules for each definition
# 5) Save comments around modules and always blocks
# 6) Convert $display statements
# 7) Make a special altera_verilog library that overloads verilog operators
30 #     That will make for a much cleaner compiler.
#
#####
#
sub V2VHD

```

```

35 {
    my ($Verilog_String,
        $complete_filename,
        $pass,
        $Module_Indexed_Port_Width_Pointer,
        $Module_Indexed_Port_Names_Pointer,
        $Module_Indexed_Parameter_List_Pointer,
        $Component_List_Pointer) = @_;
40

```

```

45 my %Entity_And_Architecture;
my %Module_Instantiation_List;
my @Module_Array;

#Put the whole shebang including `included files in $line.
$complete_filename =~ tr|\\|\\/|;
50 my @tmp_path = split (/\\//,$complete_filename);
my $filename = pop (@tmp_path);
my $path = join ("\\",@tmp_path);
#warn "path,filename $path,$filename\\n";
my $line = $Verilog_String;
55 $line .= &read_file('fh000',$filename,$path);

```

```

my $all_entity_declarations;
my $all_architecture_blocks;
my %DefParam_List;
60 my $vhdl_module = "--$complete_filename\\n\\n";

#Find and process comments
$line = &Kill_Comments($line);

```

0930106 "051201
T.021960


```

# $line = &Process_Comments($line);

# Find definitions and process ifdefs
$line = &Process_Verilog_Directives($line);

5   #warn "at beginning, line is $line\n";

my %attribute_type;
&Get_Attribute_Types (\%attribute_type);

10  #Process each module

while ($line
s/^(.*?)\bmodule\s+(\w+).*\;;(.*?)\bendmodule\b(.*)$/\1$4/s)
15  {
    my $module_name;
    $module_name = $2;
    #warn "module $module_name, pass $pass\n";
    if ($pass == 2)
    20  {
        push (@Module_Array, $module_name);
    }

    undef %Parameter_List;
    my %Parameter_List;
    my @parameter_order = ();
    my $module_commands = $3;
    my $Process_Statements;
    my $Wire_Assignments = "";
    30  my %Declared_Components;

    my $instantiation_string;
    my $attribute_string;

    35  my %attribute_already_declared;

    #Clean up module_name
    $module_name =~ s/^(.*)\s*/\1/;
    #warn "module name is $module_name\n";
    40  warn "VERILOG TO VHDL CONVERSION: ".$date_time." PROCESSING MODULE
    $module_name\n";

    #####
    # We need to know all defparams before we instantiate a module
    45  # So we look for defparams separately and store them in
    # %DefParam_List which is indexed by instance name.
    # When it comes time to instantiate a module, we will have all
    # the information we need.
    while ($module_commands =~ s/^(.*?)\bdefparam\s+(.*?)\;;(.*)$/\1$3/s)
    50  {
        #defparam instance_name.parameter = value,
        # instance_nameX.parameterY = valueZ ;
        #print "found defparam $2\n";
        foreach $defparam (split (/\\s*,\\s*/s, $2))
    55  {
            if ($defparam =~ /^\\s*(\\w+)\\. (\\w+)\\s*=\\s*(\\S+)/)
            {
                $DefParam_List{$1} .= ",\n" if ($DefParam_List{$1});
                $DefParam_List{$1} .= "    $2 => $3";
                #warn "adding $2 => $3 to dplist $1\n";
            }
            else
            {

```

09880106-061201

```

        die "ERROR: defparam statement $defparam not understood!";
    }
}

5
#####
# We need to find all exemplar attributes separately of the
# commands that are split by semi-colon.
# So we look for exemplar attributes separately and store them
10 # When it comes time to instantiate a module, we will have
# the attribute information that we need.
# warn "before exemplar check, $module_commands\n---\n";

# NO EXEMPLAR ATTRIBUTES ARE RESPECTED, MODULES WITH NO CONTENTS WILL
15 # BE CONVERTED TO BLACK BOXES
while (0) # ($module_commands =~ s/^(.*?)\-\-\-
\s*exemplar\s+attribute\s+(.*?)\s*/$1/mi)
{
    my $attribute_info = $2;
    my ($name,$attribute,$attribute_value) = split (/s+/, $2);
    $attribute =~ tr/A-Z/a-z/;
    my $att_type = $attribute_type{$attribute};
    #warn "attribute found: name,attribute,attribute_value =
20 $name,$attribute,$attribute_value,$att_type\n";
    if ($att_type)
    {
        $attribute_string .= "        --attribute $attribute : $att_type;\n"
        unless ($attribute_already_declared{$attribute});
        $attribute_already_declared{$attribute}++;
30
        $attribute_string .= "        --attribute $attribute of $name:ENTITY
is $attribute_value;\n";
    }
}
35 #warn "after exemplar check, $module_commands\n---\n";

#All other commands are separated by ";" and are processed inline with
each command.

40 #####
# Port_List only contains information for module ports
# Signal_List only contains information for wires and registers
#
# Port_Width, Port_Type contains info for all ports, wires and registers
45 # They should probably be renamed All_Nodes_Width/Type or something
#
# Port_Type is indexed by verilog (input,output,inout,reg,wire) its
result is
# a "," separated list of all ports of that type
50 #
# Port_Width is indexed by the port name. Its result is "left,right"
where
# left is the first dimension of the array, right is the last dimension.
# eg. for wire [7:3] foo; $Port_Width{"foo"} = "7,3";
55 #
# Port_List is indexed by the module port_name.
# Its value is SIGNAL $port_name : (IN|OUT|INOUT) STD_LOGIC(_VECTOR?)
(left DOWNT0 right)
#
60 # Signal_List is indexed by the register/wire name
# Its value is $Signal_List{$port} = "SIGNAL $port : STD_LOGIC(_VECTOR?)
(left DOWNT0 right)"

```

FOOTNOTES

090307.06 "061201

```
undef %Signal_List;
my %Signal_List;

5   undef %Type_List;
    my %Type_List;

    undef %Port_List;
    undef %Port_Width;
    undef %Port_Type;

10   my %Port_List;
    my %Port_Width;
    my %Port_Type;
    if (0) #($pass == 2)
15   {
        my $Port_Width_Pointer =
    $$Module_Indexed_Port_Width_Pointer{$module_name};
        die "ERROR MODULE ($module_name) HAS NOT BEEN PREVIOUSLY SCANNED\n"
        if ($Port_Width_Pointer eq "");
20   foreach $key (sort (keys(%{$Port_Width_Pointer})))
        {
            #warn "module_name ($module_name) key ($key)\n";
            $Port_Width{$key} = "Taken\n";
        }
25   }
    $Module_Indexed_Port_Names_Pointer,
    $Module_Indexed_Parameter_List_Pointer,

    my @Module_Info = (%Port_Width,%Signal_List,%Wire_Assignments);
    my $counter = 0;
    my $number_of_commands = 25;
    while ($module_commands =~ s/^(.*?)\;(.*?)$2/s)
    {
        #Counter prints "." after $number_of_commands
        $counter++;
        print STDERR "."
        if (($counter % $number_of_commands) == 0);

        my $this_command = $1;
        my $next_commands = $2;
        my $rest_of_module = "$1\;$2";
        #warn "rom is $rest_of_module \n";
        #warn "tc, $this_command\n";
        #Search for parameters and determine their type,
45   #Types are "NATURAL" if they only contain numbers, else Type is
    "STRING"
        if ($this_command =~ /^(.*?)\bparameter\s+(.*?)\s*$/s)
        {
50   my $parameter_equation;
            $parameter_equation = $2;
            my ($param_lhs,@param_rhs) = split
    (/s*\s*=\s*/,$parameter_equation);
            my ($param_rhs) = join ("=",@param_rhs);
            my ($parameter_type) = "STRING";
55   $parameter_type = "NATURAL"
                if ($param_rhs =~ s/^\s*(\d+)\s*/$1/s);
            #warn "$param_lhs,$param_rhs,pt is $parameter_type\n";
            #was $Parameter_String .= "      $param_lhs : $parameter_type :=
    $param_rhs\n";
60   $Parameter_List{$param_lhs} = "$parameter_type := $param_rhs";
            #warn "found paramter ".$Parameter_List{$param_lhs}."\n";
        }
    }
```

```
#####
# Get Signal Definitions
                                if ($this_command ==~
/(.*)\b(input|output|inout|reg|wire|integer)\b(.*)/s)
{
    my $direction = $2;
    my $port_list = $3;
    my $left_index = "";
    my $right_index = "";
    my $width;

    #####
    # (Signal Widths Width > 1) => std_logic_vector
    if ($port_list =~ s/^\s*([^\:]+\:([^\]]+)\s*(.*)/$3/)
    {
        $left_index = eval($1);
        $right_index = eval($2);

        $type =
"STD_LOGIC_VECTOR".&Vector_Order($left_index,$right_index);
        $width = "$left_index,$right_index";
    }
    else #Width is one.
    {
        if ($direction =~ /^inout$/)
        {
            #all inouts of width one should be considered
            #std_logic_vector.
            $type = "STD_LOGIC_VECTOR (0 DOWNT0 0)";
            $width = "0,0";
        }
        else
        {
            if ($direction =~ /^integer$/i)
            {
                #Convert integers to 32 bit STD_LOGIC_VECTORS
                $width = "31,0";
                $type = "STD_LOGIC_VECTOR(31 DOWNT0 0)";
            }
            else
            {
                #####
                # everything else is STD_LOGIC.
                # (wires and registers which will
                # be converted back to STD_LOGIC_VECTOR in the if ($dir)
condition below.)

                # Perhaps this could be cleaned up later
                $type = "STD_LOGIC";
                $width = "0,0";
            }
        }
    }
}

#####
# Wires can be declared and assigned to in the same statement
# e.g. wire a = b + c;
# So we strip away everything after the first "=" assignment
# And put the results in $Wire_Assignments
my $rhs;
if ($direction eq "wire")
{
    my @rhs = ();
    ($port_list,@rhs) = split (/^\s*={1}\s*/s,$port_list);
```

```

    $rhs = join ("=",@rhs); #Put $rhs back together again
}

#####
5 # Handle memories e.g.
# reg [7:0] mem_array [512 - 1 : 0];
if ($port_list =~ s/^(.*?)\[ (.*?)\:(.*?)\]\s*$/1/s)
{
10     my $up_index = eval($2);
    my $down_index = eval($3);
    $width .= ", $up_index, $down_index";
    my $array_order = &Vector_Order($up_index, $down_index);
    my $mem_type = "memory_type_$array_order";
    while ($mem_type =~ s/\s/_/){;}
15     my $memory_type =
&Get_Exclusive_VHDL_Name($mem_type, \%Port_Width);
    $Type_List{$memory_type} = "TYPE $memory_type IS ARRAY
$array_order of $type;";
    $type = $memory_type;
20 }

#####
# Get Port Names and transform names to Port/Signal_List
$port_list =~ s/^\s*(.*?)\s*$/1/s; #Strip spaces at either end
foreach $port (split(/\s*\s/, $port_list))
{
25     die "ILLEGAL PORT NAME $port, $1\n" if ($port =~ /\W/);
    $Port_Type{$direction} .= "$port,";
    $Port_Width{$port} = $width;

30     #####
    # If direction is a port, put it in port_list

    if ( ($direction eq "input") ||
        ($direction eq "output") ||
        ($direction eq "inout")
        )
    {
40         my %dir_xform;
        $dir_xform{"input"} = "IN";
        $dir_xform{"output"} = "OUT";
        $dir_xform{"inout"} = "INOUT";
        $dir_xform{"reg"} = "";
        $dir_xform{"wire"} = "";

45         my $dir = $dir_xform{$direction};
        #warn "adding SIGNAL $port : $dir $type\n"
        #if ($direction eq "output");
        $Port_List{$port} = "SIGNAL $port : $dir $type";
50     }

    #####
    # If direction is an internal signal or (an output which must
55     have a tmp
    # signal associated with it) or (an input port of STD_LOGIC)
    # declare it in %Signal_List

    if ( ($direction eq "wire") ||
        ($direction eq "reg") ||
        ($direction eq "integer") ||
        ($direction eq "output") ||
        ( ($direction eq "input") && ($type eq "STD_LOGIC") )
        )
60

```

09080106 "051201
T02190

```

(
    # Convert all STD_LOGIC to STD_LOGIC_VECTOR(0 DOWNT0 0)
    my $tmp_type = $type;
5    $tmp_type =~ s/^STD_LOGIC$/STD_LOGIC_VECTOR(0 DOWNT0 0)/i;

    #####
    # Do not put a wire/register in signal list if it has already
    # been declared in port list.
10    $Signal_List{$port} = "SIGNAL $port : $tmp_type;"
        unless $Port_List{$port};

    #wire foo = some value; // Put (foo = some value) in wire
15    assignments
        if ( ($direction eq "wire") && ($rhs ne "") )
        {
            my $wire_assignment = &V2VHD_Equation_Wrapper ("$port =
20    $rhs",@Module_Info);
            $Wire_Assignments .= " $wire_assignment;\n";
        }
        if ( $direction eq "input" || $direction eq "output")
        {
25    #####
            # VHDL will not allow you to make equations
            # with outputs in the equation rhs. So we
            # make a wire called tmp_$port for each output named
            $port.
30    # later we assign the output $port <= tmp_$port and
            change all
            # assignments in the module to use
            # tmp_$port instead of port.

35    my $tmp_port =
            &Get_Exclusive_VHDL_Name("tmp_$port",\%Port_Width);
            #keep $port as the key so we can wire up tmp_$port and
            $port later
            $Signal_List{$port} = "SIGNAL $tmp_port : $tmp_type;";
40    #warn "added ($Signal_List{$port}) in addition to
            portlist ($Port_List{$port})\n";
        }
    }
45    }
    #next if ($pass == 1);
    #cannot do next until we've determined that the entity is a black box

    if ($this_command =~ /^(s*)\bassign\b(.*?)\s*(.*)$/s)
50    {
        my $wire_assignment = " ".&V2VHD_Equation_Wrapper ("$2 =
            $3",@Module_Info)."\n";
        $Wire_Assignments .= $wire_assignment;
    }

55    #get module instantiation
    if ($this_command =~ /(.*?)\s+(\w+)\s+(\w+)\s*\(((.*)\))\s*$/s)
    {
        my $module_being_instantiated = $2;
        my $instance_name = $3;
        my $Connections = $4;
60    #warn
            "connecting
            ($module_being_instantiated,$instance_name,$Connections)\n";
    }
}

```

```

#####
# hack a register into syn_dpram
if ($module_being_instantiated =~ /^syn_dpram_(\d+)x(\d+)\_ruwr$/i)
{
5   #warn "warning, found syn_dpram\n";
    my $WrAddress = $Connections;
    my $WrClock = $Connections;
    #Orion, make a subroutine that returns what is connected to
what
10   die "WrAddress not found for LPM_ROM $instance_name"
      unless ($WrAddress =~
s/^.*?\.\s*WrAddress\s*(?!\s*(\w+).*$/\1/s);
      die "WrClock not found for LPM_ROM $instance_name"
      unless ($WrClock =~
15  s/^.*?\.\s*WrClock\s*(?!\s*(\w+).*$/\1/s);
      my $Delayed_Address = &Add_Intermediate_Signal
("dl_$WrAddress",
20  &Width_Of($WrAddress,\%Port_Width),
      @Module_Info);

      $Connections =~ s/\b$WrAddress\b/$Delayed_Address/;

      $Process_Statements .= "--GENMEM WARNING!!! SUPER HACK MADE FOR
25  VERSION 1.0 SIMULATION WARNING!!! WARNING!!!\n";
      $Process_Statements .= "--GENMEM of
      $module_being_instantiated\ .vhd DOES NOT CORRECTLY LATCH WrAddress\n";
      $Process_Statements .= "--SO WE HACK IN A LATCH HERE AND WIRE
      $module_being_instantiated\ .WrAddress to $Delayed_Address\n";
30  $Process_Statements .= "PROCESS\n BEGIN\n WAIT UNTIL
      $WrClock = \"1\";\n $Delayed_Address <= $WrAddress;\n";
      $Process_Statements .= "END PROCESS;\n";
    }
    #warn "VERILOG TO VHDL CONVERSION: ".&date_time.: " INSTANTIATING
35  $instance_name IN MODULE $module_name \n";
    #If module_being_instantiated is a black box, declare it as a
    component,
    #else instantiate it normally

40    if (1)##$Component_List_Pointer{$module_being_instantiated})
    {
      #warn "$module_name instantiates black box
      $module_being_instantiated\n";
      #warn "here is
45  component_list".($$Component_List_Pointer{$module_being_instantiated})."\n";
      $Declared_Components{$module_being_instantiated}++;
      $instantiation_string .= "$instance_name :
      $module_being_instantiated\n";
    }
    else
    {
50      if ($pass == 2)
      {
        $Module_Instantiation_List{$module_name} .=
55  "$module_being_instantiated,"
        unless ($Module_Instantiation_List{$module_name} =~
        /\b$module_being_instantiated\,/);
        #warn "module ($module_name), MIL is now
        ($Module_Instantiation_List{$module_name})\n";
60      }

      $instantiation_string .= "$instance_name : ENTITY
      work.$module_being_instantiated\ (behavior)\n";

```

```

    }

    if ($DefParam_List($instance_name))
    {
5      $instantiation_string      .=      "      GENERIC      MAP
      \(\n".$DefParam_List($instance_name)."      \)\n";
      #warn      "PARAMETERS      in
      $instance_name\n".$DefParam_List($instance_name)."\"n";
    }
10
      #####
      # I initially thought it would be really easy to just hook up the
ports. 95% of the time it is total cake.
      # But 5% of the time is a royal pain.
15      #
      # Here are the major differences between verilog and VHDL port
instantiation.
      #
      # 1) Verilog does not care that you leave a port unconnected. VHDL
20 does.
      # 2) Verilog does not care that the widths of the ports and the
widths of the connected signal
      # do not match. VHDL does.
      # 3) Verilog considers a bit vector of width 0 and a single bit to
25 be interchangeable.
      #      VHDL considers STD_LOGIC different than STD_LOGIC_VECTOR(0
DOWNTO 0).
      #
      # Problem 1 is easy to solve: Keep a list of module ports and
30 instantiated connections
      # Each Port that does not have an associated connection gets
wired to "open".
      #
      # Problem 2 is solved in the following manner:
35      # If a port is an input, e.g. ".input_signal ({~a , {4{b}}})",
      # use the results from our V2VHD_Equation_Wrapper(input_signal =
({~a , {4{b}}})...). V2VHD_Equation
      # is sophisticated enough to deal with all verilog operators and
can also reconcile widths.
40      #
      # For each output connection, e.g. ".output_signal ({~a ,
{4{b}}})"
      # We make a tmp_output_signal which has the same width as the
output port. Then we use V2VHD_Equation
45      # again, &V2VHD_Equation_Wrapper("({~a , {4{b}}})" =
tmp_output_signal"...)).
      #
      # Problem 3 is also solved by having a tmp signal act as the go-
between.
50      #
      # This makes for very ugly vhdl code. When I have time, I'll fix
it so that the
      # only time we generate a tmp signal is for the nasty 5% of the
time.
55

      my      $Module_Port_Names_Pointer      =
      $$Module_Indexed_Port_Names_Pointer($module_being_instantiated);
      die "ERROR IN INSTANTIATING ($module_being_instantiated). UNKNOWN
60 MODULE\n"
      if ( ($Module_Port_Names_Pointer eq "") && ($pass == 2));
      my      $Module_Port_Width_Pointer      =
      $$Module_Indexed_Port_Width_Pointer($module_being_instantiated);

```


09860106"061201

```
#my $Module_Parameter_Pointer =
$Module_Indexed_Parameter_List_Pointer($module_being_instantiated);

my %Connection_List;

5   foreach $connection (split (/\\s*\\,\\s*/s,$Connections))
    {
        #last if ($pass == 1);

10      if ($connection =~ /\\.\\s*(\\w+)(\\.\\s*)/s)
        {
            my $Module_Port_Name;
            $Module_Port_Name = $1;
            #my $Connection_rhs = $2;
15      my ($tmp,$Connection_rhs,$tmp2) = &Count_Parentheses($2);

            die "ERROR INSTANTIATION OF $instance_name IN MODULE
$module_name NOT UNDERSTOOD ($connection)$1\\n"
                if ($Connection_rhs eq "");
20      $Connection_rhs =~ s/^\\s*(\\.\\s*)\\s*/$1/;
            $Connection_List{$Module_Port_Name} = $Connection_rhs;
            #warn "INSTANTIATION ($module_being_instantiated) adding
            $Connection_rhs,                                     to
            $Module_Port_Name(".$Connection_List{$Module_Port_Name})."\\n";
25      die "ERROR INSTANTIATION OF MODULE $module_being_instantiated
            NAMED $instance_name \\n"
                ." IN MODULE $module_name REFERS TO AN UNKNOWN PORT
            $Module_Port_Name\\n"
                unless (($Module_Port_Width_Pointer{$Module_Port_Name}
30      || ($pass == 1));
        }
        else
        {
            die "ERROR INSTANTIATION OF $instance_name IN MODULE
35      $module_name NOT UNDERSTOOD ($connection)$1\\n";
            # $module_port_list .= "      $connection,\\n";
        }
    }

40      my $module_port_list;
      my $pw_array = join ("\\n ",keys(%Port_Width));

      foreach $port (sort(keys(%$Module_Port_Names_Pointer)))
      {
45          my $what_port_connects_to = $Connection_List{$port};
          my $port_width = &Width_Of($what_port_connects_to,
                                  $Module_Port_Width_Pointer);

          if ($what_port_connects_to eq "")
50          {
              $module_port_list .= "      $port => open,\\n";
              next;
          }
          my ($tmp_name,
55              $direction,
              $port_type)
              =
              &Get_Port_Name_Direction_And_Type($$Module_Port_Names_Pointer{$port});

          my %E_List;

60          my $port_name
              =
              &V2VHD_Equation($what_port_connects_to,@Module_Info,\\%E_List);
          $port_name =~ s/^\\s*(\\.\\s*)\\s*/$1/s;
```

090801061201
TOT190"061201

```
my $connection_width = &Width_Of($port_name,\%Port_Width);

    if ( $connection_width eq "" && ($direction =~ /^OUT$/)) &&
($pass != 1))
5      {
        warn "WARNING:  INSTANTIATION OF $instance_name IN MODULE
$module_name HAS A PORT NAMED\n";
        warn "          ($what_port_connects_to) THAT HAS NOT BEEN
DEFINED.  ASSUMING A SIGNAL OF WIDTH\n";
10      warn "          ($port_width) EQUIVALENT TO WIDTH OF MODULE
($module_being_instantiated) PORT\n";
        warn "          ($port_name)\n";
        $connection_width = $port_width;

15      &Add_Intermediate_Signal($what_port_connects_to,$connection_width,@Module
_Info);
    }

    if ($E_List($what_port_connects_to))
20    {
        #warn "e_list ($what_port_connects_to) known\n";
        $port_width = &Width_Of($what_port_connects_to,\%Port_Width);
    }

25    #If the port width does not match or if $what_port_connects_to
contains some funky operators
    #do the safe thing and make a tmp_wire.
        if (($connection_width != $port_width) ||
($what_port_connects_to =~ /\W/))
30    {
        if ($direction =~ /^INOUT$/i)
        {
            #warn (%Port_Width);
            die ("ERROR MODULE $module_name,  INSTANTIATION
35 $instance_name INOUT PORT, \n"
                ."$what_port_connects_to (width $connection_width)
MUST HAVE THE SAME WIDTH AS $port (width $port_width)\n");
        }

40        if ($direction =~ /^IN$/i)
        {
            $what_port_connects_to = "To_$instance_name\_$_port";
            $what_port_connects_to =
&Get_Exclusive_VHDL_Name($what_port_connects_to,\%Port_Width);
45            #warn "instance_name is $instance_name, port is $port,
mpwn is ".$$Module_Port_Width_Pointer{$port}."\n";
            $Port_Width{$what_port_connects_to} =
$$Module_Port_Width_Pointer{$port};
            $Signal_List{$what_port_connects_to} = &Declare_Signal
50 ($what_port_connects_to,\%Port_Width);

            $Wire_Assignments .= "      ".$V2VHD_Equation_Wrapper
("$what_port_connects_to = ".$Connection_List{$port},

55 @Module_Info)."\;\n"
                                if ($pass == 2);
        }
        if ($direction =~ /^OUT$/i)
60    {
            $what_port_connects_to = "From_$instance_name\_$_port";
            $what_port_connects_to =
&Get_Exclusive_VHDL_Name($what_port_connects_to,\%Port_Width);
```

```

                                $Port_Width{$what_port_connects_to}      =
$$Module_Port_Width_Pointer{$port};
                                $Signal_List{$what_port_connects_to} = &Declare_Signal
($what_port_connects_to,\%Port_Width);
5
                                $Wire_Assignments .= "      ".&V2VHD_Equation_Wrapper
($Connection_List{$port}." = $what_port_connects_to" ,
10
@Module_Info)."\;\n"
                                if ($pass == 2);
                                }
                                }
                                $what_port_connects_to .= "(0)" if ($port_type ==~
/^STD_LOGIC$/i);
15
                                $module_port_list .= "      $port ==>
$what_port_connects_to,\n";
                                }
                                #Lose the last comma
                                $module_port_list =~ s/\\,\s*$//s;
20
                                $instantiation_string .= "  PORT MAP \(\n";
                                $instantiation_string .= "$module_port_list    \)\;\n\n";
                                }
25
#####
# In verilog, you often declare a bunch of wire and assign statements
# right before you do an always or initial block.  It's nice to do it
this way
30
# so that wires that determine the outcome of the always statement
# are near the actual always statement.
#
# We do the same thing in our initial and always blocks.
# We put all previously assigned wires
# before the PROCESS statement.  And clear out $Wire_Assignments so
35
# that the assignments only show up in one place.
#
# find initial always statements.
#
# assume that a statement always @(posedge a or posedge b)
# always has "a" as the clock and "b" as the asynchronous event.
40
# we can get fancier later.

if ($rest_of_module =~ /^\\s*\\b(always|initial)\\b\\s*(.*)$/s)
{
45
    #update heartbeat
    print STDERR ".";
    $counter = $number_of_commands;

    #warn "in always statement\n";
    my $always_or_initial = $1;
    my $always_statement = $2;
    #warn "found always statement, $always_statement\n";
    my $clk;
    my $clk_level = "0";
55
    my $edge;
    my $asynchronous_event;
    my $asynchronous_level = "0";
    my $asynchronous_edge;
    my $wait_statement;

    my $tmp;
    my $always_condition;
    my $always_innards = $2;
60

```

```

my $tmp_always_condition;

$Process_Statements .= $Wire_Assignments;
$Wire_Assignments = "";

#Search for always @(foo)
if ($always_statement =~ /\^@(.*?)$/s)
{
    #warn "found \@ remaining $1\n";
    ($tmp,$always_condition,$always_innards) =
Count_Parentheses($1);

    #convert always condition
    $tmp_always_condition = $always_condition;

    #get clock and clocks edge
    if ($always_condition =~ s/(pos|neg)edge\s+(\S+)(.*)/$3/s)
    {
        $clk = $2;
        $edge = $1;
        $clk_level = "1" if ($edge eq "pos");
        $swait_statement = "UNTIL $clk = \'".($edge eq "pos")."\'";
        $swait_statement = "UNTIL $clk = \'" . ($edge eq "pos")."\'";

        #warn "always condition is ($always_condition)\n";
    }
    else
    {
        if ($always_or_initial =~ /always/i)
        {
            #No clock statement, but possibly a conditional always
            #e.g. always @(a or b or c)
            while ($always_condition =~ s/\s*\bor\b/,/,i){;}
            $swait_statement = "ON $always_condition";
            #warn "wait_statement is $swait_statement\n";
        }
    }
}

my $rest_of_module_innards;
($always_innards,$rest_of_module_innards) =
&Handle_Next_If_Else_Line($always_innards,

@Module_Info);

my %Variable_Conversion_List;
while ($always_innards =~ s/Please Convert (\w+) To A Variable//s)
{
    #warn "found conversion of ($1)\n";
    $Variable_Conversion_List{$1}++;
}

foreach $VCL (keys(%Variable_Conversion_List))
{
    #warn "VCL is ($VCL), width of is
".&Width_Of($VCL,\%Port_Width)."\n";
    my $tmp_name = &Add_Intermediate_Signal("VARIABLE_$VCL",
&Width_Of($VCL,\%Port_Width),
@Module_Info);
    &Convert_Signals_To_Shared_Variable($tmp_name,\%Signal_List);

    while ($always_innards =~ s/\b$VCL\b/$tmp_name/si){;}
}

```

```

                                $always_innards = "          $tmp_name := $VCL;\n$always_innards\n
$VCL <= $tmp_name;";
                                }

5          #####
          # &Handle_Next_If_Else_Line has converted everything inside the
always statement to always_innards and the rest
          # of the module is in $rest_of_module_innards; So set our top
level module_commands = $rest_of_module_innards
10          # and continue parsing from there.
          $module_commands = $rest_of_module_innards;

          #get asynchronous signal and edge if it exists
          if ($always_condition =~ /\s+or\s+(pos|neg)edge\s+(\S+)/s)
15          {
              $asynchronous_edge = $1;
              $asynchronous_event = $2;
              $asynchronous_level = "1" if ($asynchronous_edge eq "pos");

20              #my $first_if_then_statement = "IF $asynchronous_event =
              \'$asynchronous_level\' THEN";
              my $first_if_then_statement = "IF $asynchronous_event =
              \'$asynchronous_level\' THEN";

25              my $death_string = "ERROR ALWAYS CONDITION
              $tmp_always_condition HAS ASYNCHRONOUS SIGNAL ($asynchronous_event)\n"
              ."BUT DOES NOT HAVE THE SIGNAL IN THE FIRST IF STATEMENT.\n"
              ."INSIDE ALWAYS BLOCK IS ($always_innards).\n";

30              #Process first if condition
              #make sure asynchronous edge was involved in first if
              computation.

              die (" $death_string") unless ($always_innards =~
35              s/^(.*)IF(.*?)THEN(.*)$/ $1$first_if_then_statement$3/s);
              my $fic = $2;
              die (" $death_string") unless ($fic =~ /$asynchronous_event/s);
              die (" $death_string") unless
              ($always_innards
40              s/($first_if_then_statement.*?)ELSE(.*)$/ $1ELSIF $clk\'EVENT AND $clk =
              \'$clk_level\' THEN$2/s);

              $Process_Statements .= "PROCESS ($clk,$asynchronous_event)\n
              BEGIN";
45              $Process_Statements .= "          ";
              }
              else
              {
                  $Process_Statements .= "PROCESS\n BEGIN\n";
                  $Process_Statements .= "          WAIT $wait_statement;\n"
50                  if ($wait_statement);
                  #$Process_Statements .= "$always_innards \n END PROCESS;\n";
              }

55              $Process_Statements .= $always_innards;
              $Process_Statements .= "\n WAIT;";
              if ($always_or_initial =~ /initial/);
              $Process_Statements .= "\nEND PROCESS;\n\n";
              } #Done with always @innards.

60          }
          #If we've printed status ".", print a new line
          print STDERR "\n"
          if ($counter >= $number_of_commands);

```

```

#####
# Put it all together.
#
5 # Check to see if anything needs to be put inside the architecture block.
# If nothing does, assume its a black box and instantiate a "component"
with noopt
# attributes and the exact same Parameters and Ports as the entity hooked
up to the
10 # component.
# P.S. I hate black boxes.
if
(1) # (($Process_Statements.$Wire_Assignments.$instantiation_string.$attribute_string) eq "")
15 {
my $Component_String =
&Declare_Entity($module_name,\%Port_List,\%Parameter_List);
while ($Component_String =~ s/ENTITY/COMPONENT/){;}
while ($Component_String =~ s/END\s+$module_name\s*/;/END
20 COMPONENT/;){;}
my $tmp_cs = $Component_String;
$Component_String .= " --attribute noopt: boolean;\n";
$Component_String .= " attribute noopt of $module_name: component is
true;\n";
25 $Component_String .= " --Hard instantiation of $module_name
megafunction in VHDL with user defined parameters\n\n\n";
$$Component_List_Pointer{$module_name} = $Component_String;
}
#else #It is not a black box, declare it as a normal module
30 if
(($Process_Statements.$Wire_Assignments.$instantiation_string.$attribute_string) ne "")
{
#####
35 # Unlike Verilog, VHDL requires that all entities be defined before
another
# architecture block instantiates said entity. (Sounds like a legal
verdict doesn't it?)
# So, we put our port and parameter information into
40 $entity_declaration, then put all
# $entity_declaration at the top of our file.

my $entity_declaration =
45 &Declare_Entity($module_name,\%Port_List,\%Parameter_List);

#Everything below goes in the architecture block.
#I'll put a commented out entity declaration in there too so
#it will be easier for a coder to figure out what is going on
#Put it all in a string called $architecture_block
50 my $architecture_block = "ARCHITECTURE behavior OF $module_name
IS\n";
my (@Component_Array) = sort (keys (%Declared_Components));
$architecture_block .= "attribute noopt: boolean;\n" if
55 @Component_Array;
foreach $key (@Component_Array)
{
$architecture_block .= ($$Component_List_Pointer{$key});
}
60 foreach $type (sort(keys(%Type_List)))
{
$architecture_block .= " ".$Type_List{$type}."\n";
}
}

```

```

    }

    while ($Wire_Assignments =~ s/\`//g){;} #crush tick escape character
    should already be done, but it does not hurt anything
5      #warn "wa is $Wire_Assignments\n";

    #####
    # VHDL will not allow you to make equations
    # with outputs in the equation rhs. So we
10    # make a wire called tmp_$port for each output
    # and assign the output $port <= tmp_$port. All
    # other assignments in the module are changed to use
    # tmp_$port instead of port.
    #

15    # Also, VHDL considers std_logic to be different than
    std_logic_vector(0 downto 0). Verilog does not.
    # We solve this above by assuming everything is a std_logic_vector(0
    # a port (std_logic) and instantly convert it to tmp_port
20    (std_logic_vector(0 downto 0)) and set
    # tmp_port(0) <= port; if port is input and port <= tmp_port(0) if
    port is output.
    # Extra tricky, though is that fpga express does not like 'event on
    std_logic_vectors.
25    # we'll need to convert 'event signals back to std_logic

    foreach $port_declaration (sort(keys(%Port_List)))
    {
30        my ($port_name,$port_direction,$port_type) =
        &Get_Port_Name_Direction_And_Type($Port_List{$port_declaration});

        my $port_type_is_std_logic = 0;
        $port_type_is_std_logic = 1
35        if ($port_type =~ s/^STD_LOGIC$/STD_LOGIC_VECTOR(0 DOWNT0
        0)/i);

        my $tmp_signal_list = $Signal_List{$port_declaration};
        if (
40            ($tmp_signal_list
            /\^s*(SIGNAL|VARIABLE|SHARED\s+VARIABLE)\s*(\w+)/is
            #($port_direction =~ /\^OUT$/i) ||
            #($port_type_is_std_logic)
            )
        {
45            my $tmp_port_name = $2;
            #warn "port_name is $port_name tmp_port_name is $1,
            $tmp_port_name\n";
            #add new name to signal list if any Process,Wire, or
            instantiation signal uses it.
50            my $port_name_used = 0;
            #Orion, maybe could use /g option here if we knew the special
            variable for how many
            #times /g matched or we could next if we were here for pass 1

55            while ($Process_Statements ==
            s/\b$port_name\b/$tmp_port_name/s){$port_name_used++;}

            my $event_Process_Statements;
            #Remember, clk is first in an asynchronous reset statement.
            if ($port_name_used)
60            {
                while ($Process_Statements ==

```

```

# $1 = PROCESS
\(\
    $tmp_port_name(\s*\,\s*\w+\s*\)) # $2 = " ,
reset\) "
5      (. * ? \b) # $3 = reset
statement
    $tmp_port_name('\EVENT\s+AND\s+) # $4 =
'event and
    $tmp_port_name(\s+=\s+)\\"([01])\\" # $5 = " =
10 " $6 = new clk logic level
    (\s+. * ? \bEND\s+PROCESS) # $7 = rest
to end process
    /$1$tmp_port_name$2$3$tmp_port_name$4$tmp_port_name$5\'$6\'$7/six)
15 {
    $port_name_used = $port_name_used - 2;
    die "Incorrect \'event substitution\n"
        if ($port_name_used < 0);
}

20 while ($Process_Statements ==~
s/(\bPROCESS\s+BEGIN\s+WAIT\s+UNTIL\s+)\$tmp_port_name(\s+=\s+)\\"([01])\\"/
    $1$tmp_port_name$2\'$3\'/six)
    {$port_name_used = $port_name_used - 1;}
}

25 while ($Wire_Assignments ==~
s/\b$tmp_port_name\b/$tmp_port_name/s){$port_name_used++;}
while ($instantiation_string ==~
30 s/(\\=\\>. * ? \b)$port_name\b/$1$tmp_port_name/){$port_name_used++;}

if ($port_name_used)
{
    my $tmp_wire_assignment;
    $tmp_port_name .= "(0)" if ($port_type_is_std_logic);
35 if ($port_direction =~ /^OUT$/){$tmp_wire_assignment = "
$tmp_port_name <= $tmp_port_name;\n";}
    if ($port_direction =~ /^IN$/){$tmp_wire_assignment = "
$tmp_port_name <= $port_name;\n";}
    #die "DID NOT FIND ($port_name) IN ARCHITECTURE BLOCK
40 ($code_in_architecture_block)\n"
    #unless ($code_in_architecture_block ==~
s/(.*\[^\;\\\;\\=]*\b$tmp_port_name\b[^\;\\\;\\=]*\[^\;\\\;\\=]*.*)/$1$tmp_wire_assignment$2/s);
    $Wire_Assignments .= $tmp_wire_assignment;
}
else
{
    undef ($Signal_List{$port_name});
}
}
}

50 #####
# Now because FPGA express hates std_logic_vector 'event and wait
statements, we need to generate
# a std_logic signal to be used in all other 'event and wait
55 statements
my %std_logic_xform;
if (0) #No, I refuse to support FPGA express
{
    while ($Process_Statements ==~
60 s/\b(PROCESS\s*(\s*)
    (\s*)(\s*\,\s*\w+\s*\)) # $1 = PROCESS \(
    # $2 = clk, $3 = "
, reset\) "

```



```

statement      (.*?\b)                                # $4 = reset

\2(\'EVENT\s+AND\s+)      # $5 = 'event and
\2(\s+=\s+)\\"([01])\\"      # $6 = " = " $7 = new clk
5  logic level

(\s+.*?\bEND\s+PROCESS)      # $8 = rest to
end process

/$1$2$3$4($2(0))$5$2(0)$6\'$7\'$8/six)

10      {;}
      #my      $std_logic_name      =
      "($2(0))";#&Get_Exclusive_VHDL_Name(std_logic_$2,\%Port_Width);
      #std_logic_xform{$1} = $2;

      #1$std_logic_name$3$4$std_logic_name$5$std_logic_name$6\'$7\'$8/sixee)
15

      while      ($Process_Statements      =~
s/(\bPROCESS\s+BEGIN\s+WAIT\s+UNTIL\s+)\$tmp_port_name(\s+=\s+)\\"([01])\\"/
      $1$port_name$2\'$3\'/six)

20      {;}

      }

      foreach $signal (sort(keys(%Signal_List)))
      {
25      $architecture_block .= "      ".$Signal_List{$signal}."\n";
      }

      $architecture_block .= "BEGIN\n\n";

      $architecture_block
30 $Process_Statements.$Wire_Assignments.$instantiation_string.$attribute_string;
      $architecture_block .= "END behavior;\n\n\n";

      #last thing convert [a:b] to (a downto b)
      $architecture_block = &V2VHD_Index($architecture_block);
35

      $Entity_And_Architecture{$module_name}      =
&Get_Library_Declaration($architecture_block).$entity_declaration.$architecture
_block;

40      #all_architecture_blocks .= $architecture_block;
      #warn "defined $module_name\n";

      }
      #####
      #Now we are finished with the contents of the module.
45      #Save away valuable information for later regardless of if its a black
      box or not

      $$Module_Indexed_Port_Width_Pointer{$module_name} = \%Port_Width;
      $$Module_Indexed_Port_Names_Pointer{$module_name} = \%Port_List;
50      $$Module_Indexed_Parameter_List{$module_name} = \%Parameter_List;

      }
      #####
      # Now all is done, except that we need to order the modules
55      # VHDL is particular about the order in which things are declared.
      # Everything must be declared before it is instantiated. Fortunately, I
      # have a list of what is instantiated called %Module_Instantiation_List;

      my $return_string;

60      foreach $entity (&Order_Entities(\%Module_Instantiation_List,
      @Module_Array)

      )

```

```

    {
        $return_string .= $Entity_And_Architecture($entity);
    }

5   return ($return_string);
}

#####
#
10  # Order_Entities is a recursive function that takes a hash and an array of
    # keys.
    # It returns an array of entities in order of dependance. The first entity
    # returned
    # will not instantiate any other entities. The second entity returned will
15  # only instantiate
    # the first entity if it instantiates any entities. Likewise for the
    # third...last
    # entites. The last module returned will be the top level in the heirarchy.
    #
20  # The value of the hash is a comma separated string of entities that are
    # instantiated
    # within the hash.
    # i.e.
    # $$pInstantiation_Hash{a} = "first module instantiated in a, second module
25  # instantiated in a,
    # last module instantiated in a"
    #
    # @keys is an array of all entity names that should be ordered.
    #####
30  #

sub Order_Entities
{
    my ($pInstantiation_Hash,@keys) = @_;
    my @return_array;

    my $already_declared = "module already declared";
    #There is no way that we can confuse "module already declared" with a
    verilog module name
40  #module is a key word and no spaces are allowed in module names.

    foreach $key (@keys)
    {
        if ($$pInstantiation_Hash{$key} eq $already_declared)
45        {
            #warn "$key already declared\n";
            next;
        }
        if ($$pInstantiation_Hash{$key} ne "")
50        {
            my $value = $$pInstantiation_Hash{$key};
            $value =~ s/\,\s*$//;
            push (@return_array,
                &Order_Entities($pInstantiation_Hash,
55                split(/\s*\,\s*/s,$value)
                )
            )
        }
        #now everything on which $key is dependant has been declared
        #so its safe to declare it.
60        push (@return_array, $key);
        $$pInstantiation_Hash{$key} = $already_declared;
    }
}

```

```

        return (@return_array);
    }

sub V2VHD_Index
5  {
    my ($string) = @_;
    #If a variable has two vector indecies, the second one takes precedence.
    while ($string =~ s/(\[[^\]]*\]\s*)(\[[.*?\]])/$2/s){;}
    while ($string =~ s|(.*)\[(.*)\]|$1|s)
10  {
        my $rest = $3;
        #warn "found $2\n";
        $GLOBAL_DEBUG = 1;
        $string .= &Vector_Order(split (/s*\:\s*/s,$2));
15  $GLOBAL_DEBUG = 0;
        $string .= $rest;
    }
    return ($string);
}

20 #####
#
# Declare Entity
#
# Takes a module_name, \%Port_List, \%Parameter_List and
# returns a string that contains a vhd1 ready entity declaration
#
sub Declare_Entity
30 {
    my ($module_name,$pPort_List,$pParameter_List) = @_;

    my $entity_declaration .= "ENTITY $module_name IS\n";

    my @Parameter_Array = sort(keys(%$pParameter_List));
35  if (@Parameter_Array)
    {
        $entity_declaration .= "  GENERIC \(\n";
        foreach $parameter (@Parameter_Array)
        {
40  my $Parameter_String = " $parameter

: ".$pParameter_List{$parameter}."; \n";
        $entity_declaration .= $Parameter_String;
        }
        $entity_declaration =~ s/\\;\n$/\n/s; #Get rid of last semi-colon.
45  $entity_declaration .= "  \); \n";
    }

    my @Port_Array = sort(keys(%$pPort_List));
50  if (@Port_Array)
    {
        $entity_declaration .= "  PORT \(\n";
        foreach $port (@Port_Array)
        {
55  $tmp = $Port_List{$port}."; \n";
        # $entity_declaration .= $tmp; # $Port_List{$port}."; \n";
        $entity_declaration .= "    ".$pPort_List{$port}."; \n";
        }
        $entity_declaration =~ s/(\.*)\\;/ $1/s;
60  $entity_declaration .= "  ); \n";
    }
    $entity_declaration .= "END $module_name;\n\n";

```

```

    return($entity_declaration);
}

#####
5  #
  # Get_Library_Declaration
  #
  # Very simple for now, may get more complicated later

10 sub Get_Library_Declaration
  {
    my ($architecture_block) = @_;

    my $library_declaration =
15     "LIBRARY ieee;\n"
        . "use ieee.std_logic_1164.all;\n"
        . "use ieee.std_logic_arith.all;\n"
        . "use ieee.std_logic_unsigned.all;\n";

20     $library_declaration .= "\nlibrary std;\nuse std.textio.all;\n"
        if ($architecture_block =~ /\bwrite\(/);

    return ($library_declaration);
  }

25 sub V2VHD_Files
  {
    my ($Verilog_String,$Destination_Directory,@files) = @_;
    my (@Files_To_Synthesize);
30     my (@Do_Not_Synthesize_These);

    #warn "vs = $Verilog_String, dd is $Destination_Directory, files are
    @files\n";
    my $COPYRIGHT_NOTICE=
35     "--Copyright (C) 1991-2000 Altera Corporation
    --Any megafunction design, and related net list (encrypted or decrypted),
    --support information, device programming or simulation file, and any other
    --associated documentation or information provided by Altera or a partner
    --under Altera's Megafunction Partnership Program may be used only to
40     --program PLD devices (but not masked PLD devices) from Altera. Any other
    --use of such megafunction design, net list, support information, device
    --programming or simulation file, or any other related documentation or
    --information is prohibited for any other purpose, including, but not
    --limited to modification, reverse engineering, de-compiling, or use with
45     --any other silicon devices, unless such use is explicitly licensed under
    --a separate agreement with Altera or a megafunction partner. Title to
    --the intellectual property, including patents, copyrights, trademarks,
    --trade secrets, or maskworks, embodied in any such megafunction design,
    --net list, support information, device programming or simulation file, or
50     --any other related documentation or information provided by Altera or a
    --megafunction partner, remains with Altera, the megafunction partner, or
    --their respective licensors. No other licenses, including any licenses
    --needed under any third party's intellectual property, are provided herein.
    ";

55     my %Module_Indexed_Port_Names;
    my %Module_Indexed_Port_Width;
    my %Module_Indexed_Parameter_Values;
    my %Component_List;
60     my %Additional_Files;
    #warn "mipn pointer is ".$Module_Indexed_Port_Names."\n";
    my $Top_Wrapper_Name = "Top_Level_$Top_Level_Module_Name";

```

```

$Destination_Directory =~ tr|\\|\/|;
$Destination_Directory =~ s/^\s*(.*?)\s*$/$1/;
$Destination_Directory =~ s/^(.*)\//$1/;

5  foreach $pass (1,2)
    {
        warn "VERILOG TO VHDL CONVERSION: ".$date_time." PASS $pass\n";
        foreach $file (@files)
        {
10         $file =~ tr|\\|\/|;
            my $vhdl_file = $file;
            $vhdl_file =~ s/^\.*\/(.*?)$/$1/s; # Crush path. so that files in
            #warn "vhdl file is $vhdl_file\n";
            # different directories end up in the same place.
15         $vhdl_file =~ s/(.*)\.(.*?)$/$1/; #Crush all after last "."
            my $extension = $2;

            my $vhdl_innards;
            if ($extension =~ /^vhd/i)
20             {
                warn "VERILOG TO VHDL CONVERSION: ".$date_time."
                    " leaving vhdl file ($file) alone.\n";
            }
            else
25             {
                $vhdl_file = "$Destination_Directory\/$vhdl_file.vhd";
                warn "VERILOG TO VHDL CONVERSION: ".$date_time." SCANNING $file
\n";# TO $vhdl_file.\n";

30                 $vhdl_innards = &V2VHD($Verilog_String,
                                        $file,
                                        $pass,
                                        \%Module_Indexed_Port_Width,
                                        \%Module_Indexed_Port_Names,
35                                     \%Module_Indexed_Parameter_Values,
                                        \%Component_List);
            }

            if ($pass == 2)
40             {
                # If there is nothing in vhdl_innards copy the verilog file
                directly. It will get used later in quartus.
                # otherwise, make a vhdl file. No need to copy the verilog file if
                the output directory
45                 # is ".".
                if ($vhdl_innards =~ /^^\s*$\/s)
                {
                    #Orion, debug this.
                    if (0) #($Destination_Directory ne ".")
50                     {
                        open (SRCFILE, "< $file") || die "FILE ERROR! CAN NOT OPEN
$file $!\n";
                        my $Dest_File = "$Destination_Directory\/$file";
                        open (DESTFILE, "> $Dest_File") || die "FILE ERROR! CAN NOT
55 OPEN $Dest_File $!\n";
                        while (<SRCFILE>)
                        {
                            print DESTFILE $_;
                        }
                        close (SRCFILE);
                        close (DESTFILE);
                        print "COPIED $file TO $Destination_Directory\n";
60                     }
                }
            }
        }
    }

```

9880106 "061201
1021201

```

    }
    else
    {
        open (DESTFILE, "> $vhdl_file") || die "FILE ERROR! CAN NOT
5  OPEN $vhdl_file $!\n";
        print DESTFILE "$COPYRIGHT_NOTICE";
        print DESTFILE "$vhdl_innards";
        close (DESTFILE);
        warn "VERILOG TO VHDL CONVERSION: ".$date_time." CONVERTED
10 $file TO \n".(" "x60")."$vhdl_file\n";
        push (@Files_To_Synthesize,$vhdl_file);
    }
}
}
15 }
    return (@Files_To_Synthesize);
}

sub V_TCL2VHD_TCL
20 {
    my ($Destination_Directory,@files) = @_;
    foreach $file (@files)
    {
        open (TCL,"< $file") || die "FILE ERROR! CAN NOT OPEN $file $!\n";
25 my $tcl_file = $file;
        $tcl_file = $file;
        $tcl_file =~ s/(.*)\.tcl/$1\_vhd\.tcl/;
        $tcl_file = "$Destination_Directory\/$tcl_file";
        open (TCL_VHD,"> $tcl_file") || die "FILE ERROR! CAN NOT OPEN $tcl_file
30 $!\n";
        while (<TCL>)
        {
            chomp;
            while (s/\.v(\W)/\.vhd$1/){;}
35 s/\.tcl/_vhd.tcl/g;
            #s/^(s*set_clock\s+\.port\s+\.name\s+)(\S+)(.*)/$1$2\0$3/;
            print TCL_VHD "$_\n";
        }
        close TCL;
40 close TCL_VHD;
    }
}

# Do not forget the 1 at the end of a pm file
45 1

```

vpp.pm

#!/c/perl/bin/perl.exe

```
5 $GLOBAL_COPYRIGHT_NOTICE=<<END_OF_COPYRIGHT_STRING ;
//Copyright (C) 1991-2001 Altera Corporation
//Any megafunction design, and related net list (encrypted or decrypted),
//support information, device programming or simulation file, and any other
//associated documentation or information provided by Altera or a partner
10 //under Altera's Megafunction Partnership Program may be used only to
//program PLD devices (but not masked PLD devices) from Altera. Any other
//use of such megafunction design, net list, support information, device
//programming or simulation file, or any other related documentation or
//information is prohibited for any other purpose, including, but not
15 //limited to modification, reverse engineering, de-compiling, or use with
//any other silicon devices, unless such use is explicitly licensed under
//a separate agreement with Altera or a megafunction partner. Title to
//the intellectual property, including patents, copyrights, trademarks,
//trade secrets, or maskworks, embodied in any such megafunction design,
20 //net list, support information, device programming or simulation file, or
//any other related documentation or information provided by Altera or a
//megafunction partner, remains with Altera, the megafunction partner, or
//their respective licensors. No other licenses, including any licenses
//needed under any third party's intellectual property, are provided herein.
25 //Copying or modifying any file, or portion thereof, to which this notice
//is attached violates this copyright.
END_OF_COPYRIGHT_STRING

30

#
# Verilog Pre-processor.
35 #
# Giving you the power of the Perl language from within your
# Verilog source file.
#
# Any text between /*( - )*/ pairs will be treated as a Perl expression.
40 # You may put any Perl code in there that you like. You may define variables
# and call functions. You may even define your own Perl functions,
# then call them.
#
# ***** FILE GENERATION & USAGE *****
45 #
# In a DOS box, you invoke vpp, like any other Perl program, by typing
# "perl vpp.pl <arguments>". Here's the whole usage story:
#
# Usage:
50 # perl vpp.pl [-D <dest-dir-name>]
# [-P <output-file-name-prefix> ]
# [-L <lib-dest-file-name> ]
# [-X <forced-output-filename-extension> ]
# [<global-perl-var-name>=<var-value>] ...
55 # <source_file> [, source_file]...
#
# ** <source_file> arguments:
#
# Every source file you list will be given the /*( - )*/ perl-expansion
60 # treatment, and a corresponding pre-processed output file of the
# same name, with the prefix "GENERATED_", will be produced in the
# output directory. Unless you specifically say otherwise, the
# output directory will be "../GENERATED/"
```

```

#
# ** -D <dest-dir-name> argument (optional):
#
# As-mentioned above, vpp will automatically output the pre-processed
5 # files to the directory ./GENERATED/ by default. You can specify another
# directory using the -D argument.
#
# ** -P <dest-dir-name> argument (optional):
#
10 # For each source file processed, vpp will create (generate) a processed
# output file (emitted into the destination directory, per -D above).
# But what is the output file named? By default, it gets the name
# of the original source file, with the prefix "GENERATED_" added.
# Thus, by default, Vpp transforms the file "foo.v" into the file
15 # "GENERATED/GENERATED_foo.v". If you want some other prefix, you
# can set it with the -P argument. The prefix "-" (dash) is magic,
# and is interpreted as: "keep your stinking prefix away from my file."
#
# ** -L <lib-dest-file-name> (optional):
20 #
# The "-L" argument sets the generated "library" output file. In the
# course of doing their work, some vpp generator-functions actually
# -create- modules (&Mux does, for example). These modules need to
# live in a file somewhere. This argument tells vpp where to put
25 # automatically-generated "library" modules. This argument is
# optional. If you don't specify, all auto-generated modules will
# be emitted to a file named "generated_vpp_modules.v" in the
# destination-directory.
#
30 # ** -X <forced-output-filename-extension> (optional):
#
# If, for example, you were reading-in a bunch of files with
# the extension ".vpp" (e.g. "my_little_pony.vpp"), and you wanted
# to generate a bunch of like-named files with the extension
35 # ".v" (e.g. --> "my_little_pony.v"), you would just say "-X v" on
# the command line. I think that's about all there is to it.
# ALL output files will be coerced to have this extension.
#
# ** <global-perl-var-name>=<var-value> arguments (optional):
40 #
# Vpp is all about parametric / conditional verilog generation.
# The Vpp sources themselves contain a good deal of Perl code.
# The Perl code in the source files is often "controlled" by a
# set of global variables that can be thought of as "parameters."
45 # For example, a single Perl variable called "$DELTA_32" controls
# whether the Delta CPU core is built as a 32-bit machine ($DELTA_32=1)
# or as a 16-bit machine ($DELTA_32=0). One way to set this variable is
# to place a line at the top of the "delta.v" source file itself. This
# works, but it means you actually have to go in and edit a line in the
50 # source file to build the "other flavor" of CPU.
#
# In general and often, you want to build a piece of hardware either
# "this-a-way" or "that-a-way", as-controlled by a Perl variable
# "at the top of the file" (e.g. $WAY="this-a" or $WAY="that-a").
55 # Vpp lets you set Perl variables, visible to stuff inside your
# embedded Perl code, right on the Vpp command line. This lets
# you parametrically-generate the output by running Vpp with
# different command line args. Can I stop explaining this now? You
# must surely get it. (This is just like setting pre-processor
60 # macros on the command-line of your C-compiler).
#
# You may set as many Perl variables on the command line as you
# like, or none at all.

```



```

#
# NOTE: argment ordering -- setting variables vs. source files.
# In general, all Perl-variable-setting arguments should come -before-
# the source-file arguments.
5 #
# ***** Vpp: CONCEPTUAL INTRODUCTION *****
#
# In principle, you can start a Verilog file with "/*{", write your entire
# chip in Perl, and end it with a "*/". I've written a variety of
10 # reasonably-sophisticated (and hopefully-useful) Perl functions that manage
# much of the drudgery of Verilog coding. I've used them to create a
# working 50KGate design (~3000 equivalent Verilog lines) which is
# 95% Perl and 5% naked Verilog. But before we get to the fancy predefined
# "generator" functions, let's first cover the basics.
15 #
# ---> Perl code that GENERATES Verilog <---
#
# Your encapsulated Perl code ultimately "does something" by PRINTING into
# the OUTPUT FILE (the output file is the result of the pre-processor).
20 # Any explicit "print" statements in the embedded Perl code emit their results
# into the output file. [But, as you shall see later, directly using the
# built-in Perl "print" function is seldom necessary, and is frowned-upon].
#
# For some simple tasks (like Perl-based "`define constants"),
25 # explicitly printing things from your Perl code can be awkward. The following
# reference to the Perl variable $my_bus_width works, but is certainly awkward:
#
#     wire [(((/*{ print "$my_bus_width"; }*/) - 1) : 0] my_bus;
#
30 # In such cases it's convenient to dispense with the explicit "print"
# statement. Thus, vpp _sometimes_ automatically prints the final
# "return value" of the embedded Perl expression so you don't have to
# type "print" before every single thing you do. (I'll explain the _sometimes_
# rules later).
35 #
# Thus, the preceding example could be more compactly written as:
#
#     wire [(((/*{ $my_bus_width }*/) - 1) : 0] my_bus;
#
40 # Just as an aside, the preferred way to write that statement in vpp is:
#
#     wire /*{&W( $my_bus_width )}*/ my_bus;
#
# (The predefined "&W()" function is discussed in some detail below).
45 # Because the evaluated results of Perl expressions are emitted directly
# into the Verilog output, vpp can (if you like) supercede all the
# functionality of Verilog's native, lame 'define-macros. In particular,
# vpp allows you to compute (gasp!) constant-values with arbitrarily-complex
# mathematics. Perl (being a real programming language) supports things like
50 # logarithms, which are conspicuous by their absence in Verilog.
# Here's an example to give you a taste:
#
#     /*{
#         $ADDRESS_BITS = &Bits_To_Encode ($MAX_ADDRESS);
#     }*/
55 #     wire /*{ &W($ADDRESS_BITS) }*/ address_bus;
#
# And, yes, the "&Bits_To_Encode" function is predefined for you, and
# documented below. And, because this is all just Perl, you're free
60 # to define your own functions just like "&Bits_To_Encode" which implement
# arbitrarily-complex math, right up to cosines.
#
# ---> Five ways to emit Verilog <---

```

```

#
# Here are the five ways you can generate Verilog statements from
# embedded Perl (details follow):
#
5 # 1) Type any Perl expression. The result sometimes gets printed.
#
# 2) Use a "print" statement (discouraged).
#
# 3) Use the "&Vprint" function (better).
10 #
# 4) Use the -- (dash-dash) embedded-Verilog-literal syntax (best).
#
# 5) Call a Perl function (user-defined or pre-defined) which internally
# uses one of the above four methods.
15 #
# **** 1. Emitting Simple Perl Expressions.
#
# Often, you just want the value of a Perl variable (or expression) to
# get emitted into the pre-processed output. This is the default behavior
20 # for any embedded Perl: The ultimate "return-value" of the entire
# embedded expression gets written into the output file. Often,
# this is exactly what you want, like this:
#
# wire /*{ $my_computed_wire_name }*/;
25 #
# But consider what happens when you *set* (as opposed to *use*) a
# Perl variable:
#
# /*{ $my_computed_wire_name = $module_name . "_$i"; }*/
30 #
# In this example, I am building-up a perl variable (a name in a loop)
# that I'll use later. Unfortunately, the expression I typed does
# have a value (the string-value assigned to $my_wire_name). Because
# vpp just emits the "return value" of whatever perl expression you
35 # type, the wire name you computed gets emitted directly into the output
# Verilog file, right there where you only wanted to set a variable.
# Surely not what you wanted. Another example is even
# more heinous. Suppose you defined these constants at the top of your file:
40 #
# /*{ $data_bus_width = 16;
# $address_bus_width = 24;
# }*/
#
# The "return-value" of this embedded expression is 24, so vpp will emit
45 # a naked "24" at this point right into your Verilog output file--and
# a naked "24" (outside any module, on a line by itself) is not a valid
# Verilog statement. Instant syntax-error.
#
# So. There needs to be some way to tell vpp when you want the return-value
50 # of your expression printed, and when you don't. There are two ways. Here
# are the rules:
#
# *The return value of any Perl expression is emitted *unless*:
#
55 # 1) The embedded Perl expression started with "/*{q"
# (note the 'q', for "quiet").
#
# 2) The embedded Perl expression started with "/*{" on a line
# by itself.
60 #
# So, to fix the above heinous "data_bus_width" example, we'd just do this:
#
# /*{

```

```

#           $data_bus_width    = 16;
#           $address_bus_width = 24;
#       */
#
5  # The opening "/*{", on a line by itself, tells vpp that the following
# Perl expression is probably some sort of mini-program (lines-o-code)
# (as opposed to a quick reference to a variable), and that the
# final return-result should not automatically be printed. You are, of
# course, free to *explicitly* print things from such an embedded
10 # statement using one of the other methods 2..5, but now the choice is yours.
#
# ***** 2. Calling the "print" Perl facility directly
#
# As I've said to the point of tedium, the output of any "print" statement
15 # in the embedded Perl gets sent directly into the pre-processor output.
# Note that vpp uses funny internal Perl magic to re-direct the "print"
# output to the pre-processor output file instead of STDOUT. You, the
# user, do not need to (and should not) give a filehandle to your
# "print" statements (unless you're Perl program internally uses
20 # files for some purpose, which would certainly be twisted).
#
# If you use "print," the result will be correct but not pretty.
# The indentation of the printed text will probably be wrong, and
# there will be no indication of where the printed text came from.
25 # In other words, using "print" directly is functional, but not
# pretty. The pre-processor output file will be hard to look at if you
# use a lot of direct "print" statements. And there's not much of a reason
# to use "print" directly because vpp gives you:
#
30 # ***** 3. The built-in &Vprint function
#
# If you want to directly emit text into the pre-processor output,
# the &Vprint function formats it nicely so the output file is easier
# to look at. In particular, the text is printed with a starting left-
35 # margin set by the Perl expression's opening "/*{". This indentation
# makes a huge readability difference.
#
# Secondly, each hunk-o-text emitted by &Vprint is preceded by an
# expression-number comment. This tells you *which* Perl expression
40 # generated the resultant text. This may seem silly, but it ends up
# really saving the day. When there's a syntax error in a bunch of generated
# code (that you didn't type, and may never have seen before), the
# first question is usually "where did all this shit come from?" It's
# really handy if there's a juxtaposed comment that tells you which
45 # expression generated the code.
#
# All the original Perl expressions passed along into the pre-processor
# output (still hidden from Verilog by /*{ - }*/ comment-pairs). Each one
# is recognized (of course) and given a unique number, which happens to
50 # be the line number on which it started. That expression number
# is re-emitted as a /* - */ comment (with indentation) before every
# output from &Vprint. This makes it very easy to track-down
# the original Perl code when errors occur. That all sounds complicated,
# but it's actually pretty simple. Let me give you an example of what
55 # you'd see in the preprocessed output file if you use &Vprint:
# (This is a real example, cut-and-pasted directly from a vpp-generated
# file):
#
# /*{ --67--:
60 # # Altera gives you this one-bit "TRI" module for tri-stating
# # signals. Great, but I need 16 of them. It's a good thing I'm
# # writing this in Perl where I have loops (Iterated Verilog instances
# # don't work in Quartus).

```

```

#   for (my $i = 0; $i < $uP_DATA_BITS; $i++)
#   {
#       -- TRI Data_Bus_Buffer_$i (
#       --   .oe (do_drive_data_bus),
5   #       --   .in (uP_DATA_out [$i ]),
#       --   .out (uP_DATA      [$i ]))
#       -- );
#   } */
#   /* 67 */ TRI Data_Bus_Buffer_0 (
10  #   /* 67 */   .oe (do_drive_data_bus),
#   /* 67 */   .in (uP_DATA_out [0 ]),
#   /* 67 */   .out (uP_DATA      [0 ]))
#   /* 67 */ );
#   /* 67 */ TRI Data_Bus_Buffer_1 (
15  #   /* 67 */   .oe (do_drive_data_bus),
#   /* 67 */   .in (uP_DATA_out [1 ]),
#   /* 67 */   .out (uP_DATA      [1 ]))
#   /* 67 */ );
#
20  #   ... omitted for brevity (2 - 14) ...
#
#   /* 67 */ TRI Data_Bus_Buffer_15 (
#   /* 67 */   .oe (do_drive_data_bus),
#   /* 67 */   .in (uP_DATA_out [15 ]),
25  #   /* 67 */   .out (uP_DATA      [15 ]))
#   /* 67 */ );
#
# There you can see the original Perl expression, which has been
# given a number ("--67--:", inserted by vpp), and all the Verilog code
30  # it generated. Each line is preceded by a /* 67 */ to indicate where
# it came from, and that it was, in fact, generated.
#
# "But Wait," I hear you stridently object, "I can see the original
# Perl expression, and there's not a single call to '&Vprint' anywhere
35  # in there!" Well, yes. You're right, of course. In practice, no one
# ever actually calls &Vprint because of the glorious "--" (dash-dash)
# Verilog-literal syntax. Let me tell you all about it.
#
# **** 3. The "--" (dash-dash) Verilog-literal syntax.
40  #
# Because your embedded Perl expressions are, after all, only there
# to print Verilog, vpp gives you a super-quick shorthand for
# generating a line of verilog. Any line which starts with
# "--" (two dashes, pre- and post-whitespace ignored) is taken as a line of
45  # Verilog to be printed, after variable expansion (interpolation), into
# the pre-processor's output. This syntax is line-based: Anything
# on the line after a leading "--" will get printed, semicolon
# and all, as double-quoted text. You get a free newline, so you
# are relieved of ending all your Verilog lines with an explicit "\n".
50  # All of which adds up to a very tidy way to generate Verilog code
# from within Perl, perfectly illustrated by the above example.
#
# Now it may strike you as odd that vpp provides you with a way of
# embedding Perl within Verilog, and then builds a whole new
55  # mechanism for embedding Verilog in the Perl, which is itself embedded
# in Verilog. The obvious question: "Why not just type the stinkin'
# Verilog in the first place, and do away with the Perl altogether?" Good
# question. Easy answer: Variable-interpolation (and, to a lesser
# extent, repetition (looping)). If Perl $-variables were not interpolated
60  # into the "--" - embedded Verilog, the whole exercise would, in fact,
# be totally pointless. But, of course, Perl $-variables *are* interpolated
# (according to Perl's double-quoted-string substitution rules). And I
# shouldn't have to tell you how valuable that is. This gives

```

```

# you the power to generate fully-parametric Verilog source code.
# Verilog's lack of VHDL's ad-hoc, limited "generate" syntax is now completely
# filled -- and not with some limited, abominable kludge, but with a
# complete, real, documented, and familiar programming language.
5 #
# Just FYI, the "--" dash-dash syntax is really just a shorthand for
# passing the remainder of the line (as a double-quoted string with a
# newline stuck on the end) as an argument to &Vprint. You could,
# of course, call &Vprint explicitly, but the function name, parenthesis,
10 # and quotation marks are quite a lot of overhead, and end up being
# distracting. The results would be the same, but the "--" -trick makes
# the original source much more readable, easier to type, and harder
# to screw up.
#
15 # With what I've shown you so far, you can completely replace Verilog's
# pathetic 'define-macros, and have access to a fully-featured "generate"
# utility that kicks ass on the thing in VHDL. This is where a
# traditional pre-processor would leave you standing at the bus
# stop. But I'm going to give you the whole ride.
20 #
# ***** ADVANCED STUFF *****
#
# Vpp (and an associated file, "Generator_Functions.v") give you a
# pre-built library of Perl functions, intended to ease your Verilog
# programming task. You are free, of course, to use as much or as little
# of this library as you like. Better still, you are free to expand
# the library to fill whatever deficiencies you imagine.
25 #
# I) ELIMINATING DRUDGERY: Port lists.
30 #
# Complex Verilog designs contain a lot of redundant information. The poor
# programmer frequently has to type and re-type the same names, numbers,
# etc. without error. Designs which follow consistent coding guidelines
# are especially redundant. Let me give you an example: Some module "foo",
35 # with a few inputs and outputs, which is instantiated by "bar:"
#
#     module foo (in_1, in_2, in_3, out_1, out_2);
#         input    in_1;
#         input    [1:0] in_2;
#         input    [2:0] in_3;
#         output    out_1;
#         output    [1:0] out_2;
40 #
#         -- Contents --
#     endmodule
#
#     module bar (...bar's args...);
#         ..definition of bar's args...
50 #
#         // Instantiate "foo"
#         // But first, declare all the connection wires:
#         wire    in_1;
#         wire    [1:0] in_2;
#         wire    [2:0] in_3;
#         wire    out_1;
#         wire    [1:0] out_2;
55 #
#         foo The_Foo (
#             .in1    (in1),
#             .in2    (in2),
#             .in3    (in3),
60 #

```

09880106 "061201
TOTAL

```
#             .out1 (out1),
#             .out2 (out2)
#         );
#     endmodule
5
# Notice that you, the user, had to type "in_1" correctly, with the correct
# width, no fewer than five (5) times. And typical designs are even worse
# than this trivial example, because many of foo's ports are passed through
# bar's ports, to whomever is instantiating bar, and so on. Thus the
10 # plumbing-chore of adding another input (say, "in_4") to foo can be
# a reasonably-big hassle, and an opportunity for error.
#
# Wouldn't it be great if there were some way to state, in one neat,
# self-contained place, "here are the names of foo's ports, their widths, and
15 # directions". Then, later, refer to them as a group? Yes, it is great.
#

$VPP_PROJECT_ID_HASHCODE = '#_d-+@@!--&&n==[[](*#\$)*()*$\#\$\.ars';
$VPP_COPYRIGHT_WARNING_STRING =
20     "Warning--see copyright notice: ($VPP_PROJECT_ID_HASHCODE)";

#####
# goldfish & ribbit
#
# It would be very nice to have the Perl library functions
# "carp" and "croak", which are defined in the module "carp.pm"
#
# Sadly, including standard Perl libraries in a platform-independent
# way is tricky. Aaron's solution: Write our own "carp" and "croak"
30 # functions, and give them silly names:
sub goldfish
(
    my ($msg) = @_;
35     # Grab info about the caller of my caller.
    my($package, $filename, $line) = caller(1);
    die "$filename line $line: '$msg'\n";
}
40 sub ribbit
(
    my ($msg) = @_;
    # Grab info about the caller of my caller.
    my($package, $filename, $line) = caller(1);
45     die "$filename line $line: '$msg'\n";
}

#####
50 # Strip_Perl_Comments ($expr)
#
# Takes a string, which might be a multi-line Perl expression.
# For some reason, the Perl "eval" function is too lame to ignore
# comments embedded in the eval-expresso. Fine. We'll strip
55 # the comments so it doesn't have to.
#
#####
sub Strip_Perl_Comments
(
60     my $expr;
    my @lines;
    my $stripped_expr;
    ($expr) = (@_);
```

09:06:12.01
102190.061201

```
@lines = split (/\n/, $expr);

$stripped_expr = "";
5  foreach (@lines)
    {
        $stripped_expr .= "\n", next if /^\/#/;
        s/^(.*?)[^\]\#\.*$/$1/;
        $stripped_expr .= "$_\n";
10    }
    return $stripped_expr;
}

#####
15 # Strip_Verilog_Comments ($expr)
#
# Takes a string, which might be a multi-line Verilog expression.
# Strips out all the "//" comments.
#
20 #####
$global_strip_mode = "normal";
sub Strip_Verilog_Comments
{
    my $expr;
    ($expr) = (@_);

    my $stripped_expr;
    $stripped_expr = "";

30    my @lines;
    @lines = split (/\n/, $expr);

    foreach $line (@lines)
    {
        $line = s/^(.*?)\/\./.*$/$1/;
        $stripped_expr .= "$line\n";
35    }
    return $stripped_expr;
}

40 #####
# Expand_Literal_Verilog_Lines
#
# When you're auto-generating Verilog in a Perl statement, it's
45 # really important to be able to emit lines of Verilog code without
# turning it into a federal case.
#
# We've made it pretty easy by just declaring that anything you
# print (using "print" or &Vprint) gets emitted into the Verilog file.  Pretty
50 # easy, but not easy enough.
#
# It ends up being awkward to write a block of verilog code
# by starting every line with:
#
55 #     &Vprint("
#
# and ending every line with:
#
#     \n");
60 #
# In particular, it's easy to forget and hard to read.
#
# So, to make Verilog code generation even easier, we declare that
```

```

# any line in a Perl expression that starts with two dashes ("--")
# is a line of Verilog, and the two dashes are interpreted as a
# shorthand for "print the rest of this line into the Verilog file."
#
5 # This is accomplished by literally replacing '--' with '&Vprint(" '
# (at the beginning of lines), and ending such lines with an
# automatic '\n'); '.
#
# This definitely falls under the category of syntactical sugar. But it
10 # sure is sweet.
#
# One other nit: If a '#' character appears in the line,
# backslash it. This is because '#' is a legitimate Verilog character,
# and we later go out of our way to strip-out '#' - delimited Perl comments
15 # unless they're preceded by '\\'
#
# DANGER: Conflict with vaild Perl syntax.
#
# Mostly, in Perl programs, you don't start lines with the
20 # character string "--". Mostly. Unless, of course, you're
# pre-decrementing a value:
#
#     $i = 37;
#     while ($i)
25 #         --$i;
#
# All I can say is: Don't pre-decrement things at the beginning of lines.
# and, if you really must, I can select a different Verilog-literal
# marker sequence. Don't hold your breath.
30 #
# Orion has selected a different Verilog-literal marker sequence, but the old
# one still works as well. Because emacs doesn't tabulate comments
# (i.e. perl code) in verilog mode well. I've added understaining of
# "b(" and "e)". When alone on a line, these two get translated into "begin"
35 # and "end" respectively. Perl-mode also doesn't understand -- as perl code.
# it thinks its pre-decrement and the tabs get screwed up as a result. My
# new Verilog marker is "_ ". perl mode in emacs does the right thing with
# "_ ".
40 #
# One other hack (this is starting to get dirty):
# lines that start with "-->" don't count. If you want to emit a Verilog
# line that starts with ">", you have to say: "-- >". Note the extra space.
# We special-case "-->" because we've gone and defined a tabular-data syntax
45 # that encourages users to type this symbol. See "&Parse_Named_Arguments,"
# below.
#
#####
50 sub Expand_Literal_Verilog_Lines
{
    my $expr;
    my @lines;
    my $expanded_expr;
    ($expr) = (@_);
55
    @lines = split (/\\n/, $expr);

    $expanded_expr = "";
    my $line_number = 0;
60
    foreach (@lines)
    {
        # Special escape for dealing with lines that start with "-->"
        # (There's probably some way to do this in a single regexp,

```



```

# but I don't care):
my $is_verilog = 0;
my $do_print_into_lib_file = 0;
my $verilog_string = "";
5 if ((!/^s*-->/) && (!/^s*<--/)) # --> or <-- disqualifies it.
{
    $is_verilog = /^s*--(L?)(.*)$/;
    $do_print_into_lib_file = ($1 eq "L");
    $verilog_string = $2;
10 }

if (/^(\s*)b(\s*)\{(\s*$)/)
{
    $is_verilog = 1;
15 $verilog_string = $1.$2." begin";
}

if (/^(\s*)b(\s*)\{(\s*$)/)
{
    $is_verilog = 1;
20 $verilog_string = $1.$2." begin";
}

if (/^(\s*)\}(\s*)e\s*$/)
{
    $is_verilog = 1;
25 $verilog_string = $1.$2." end";
}

if (/^(\s*)\_(\s+)(.*)$/)
{
    $is_verilog = 1;
30 $verilog_string = $1.$2." ".$3;
}

$expanded_expr .= "$_\n", next if !$is_verilog;

# escape the '#' character, so it doesn't get stripped-out as
# a Perl comment:
40 $verilog_string =~ s/\#/\\/#/g;

# Escape the quote-character, because it foils our "surround
# the string with quotes" strategy:
$verilog_string =~ s/\"/\\/"/g;

45 # Escape the quote-character, because it foils everything
$verilog_string =~ s/\'/\\/\'/g;

    if ($do_print_into_lib_file) {
50 $expanded_expr .= "&Lprint (\"$verilog_string\\n\");\\n";
    } else {
        $expanded_expr .= "&Vprint (\"$verilog_string\\n\");\\n";
    }

55 $line_number++;
}
return $expanded_expr;
}

60 #####
# Build_Width_String
#
# If you have a wire that's 8 bits wide, you often need to get a

```

T02190" 90T08850

vpp_ptf_parse.pm

```
#####
# parse_ptf.pm
#
# A set of Perl subroutines for parsing and writing PTF-files.
#
# PTF-files used to be known as SDF-files. That's why these
# subroutines all have "SDF" in thier names.
#

#####
#
# Parse_SDF_Files
#
# This takes an array of filenames that contain SDF data. It turns them into
# an associative array so that the rest of perl can hack away. (I'm such a
# poet.)
#
# This function is used in:
#   mk_systembus.pl
#
#####
#
sub Parse_SDF_Files
{
    my $text_string = "";
    my $Top_Level = shift(@_);
    foreach $filename (@_)
    {
        # print STDERR "Parse_SDF: Opening $filename.\n";

        open (SDF,"<$filename") or die "can not open $filename: $!";
        while (<SDF>)
        {
            s/^(.*?)\#.*$/\1/;
            s/^\s+//g;
            s/\s+$//g;
            s/^\s*(.*)$/\1/;
            tr/"//";
            $text_string .= $_."\n";
        }
        close (SDF);
    }

    # Make some adjustments to normalize the text string:
    #   * Put every statement (ending in ";") on a line by itself.
    #   * Put every "{" or "}" on a line by itself.
    #
    # This makes parsing easy and natural. We do this
    # by -first- stripping-out all the newlines
    # in the file, then adding them back in where we want them.
    #
    # NOTE: PARSING LIMITATION:
    # Your NAME=VALUE -VALUES cannot contain the characters:
    #   ; { or }
    # Since VALUES are quoted, it would be possible to make
    # this parser handle these characters. Fine. You do it.
    #
    $text_string =~ s/\s+//mg;
    $text_string =~ s/\n+//mg;
    $text_string =~ s/\\;/\n/mg;
    $text_string =~ s/\\{/\n\{/mg;
```

09880106-061201

```
$text_string =~ s/\}\n\}\n/mg;
```

```
my @tree = ();  
push (@tree, "$Top_Level");  
my $tree_path = "";
```

```
undef %PARAMETER;  
my %PARAMETER;
```

```
foreach (split (/s*\n\s*/,$text_string))
```

```
{  
    $tree_path = join(':',@tree);  
    if (/^([\^\=]*)\=\\"?([\^\"]*)\\"?/)
```

```
{  
        $PARAMETER{"$tree_path:$1"} = $2;  
        $PARAMETER{"LEAF:$tree_path"} .= "$1,";  
        my $ltp = $PARAMETER{"LEAF:$tree_path"};
```

```
}  
if (/\/(\/)
```

```
{  
    $last_line =~ s/^\s*(.*?)\s*$/$1/;  
    $PARAMETER{"LEAF:$tree_path"} .= "$last_line,";  
    my $l = $PARAMETER{"LEAF:$tree_path"};
```

```
    push (@tree,$last_line);
```

```
}  
if (/\/\}\n\}\n/)
```

```
{  
    pop (@tree);
```

```
}  
$last_line = $_  
}
```

```
my $param_ptr = {};  
%$param_ptr = %PARAMETER;
```

```
return ($param_ptr);
```

```
}
```

```
#####
```

```
#  
# sub Print_SDF_File takes PARAMETER list references  
# Prints out a .sdf file
```

```
#  
# This function is used in:  
# my_systembus.pl
```

```
#####
```

```
#  
sub Print_SDF_File
```

```
{  
    my ($filename, $PARAMETER_ptr,$stop_level,$suppress_comments) = @_;  
    my %PARAMETER = %$PARAMETER_ptr;
```

```
    open (SDFOUT, "> $filename") or die "unable to open $filename, $!";  
    my $old_out = select(SDFOUT);  
    &Print_SDF_Level ($PARAMETER_ptr,$stop_level,$suppress_comments);  
    close(SDFOUT);  
    select ($old_out);
```

```
}
```

```
#####
#
# sub Print_SDF_Level takes PARAMETER list references and $level.
# Prints out a level in the sdf file.
5 # "Level" means something that has parameters and/or other levels
# underneath it. Level "foo" looks like:
#
#   foo
#   {
10 #       something="something_else"
#       something2="something_else2"
#       something3="something_else3"
#       another_level here
#       {
15 #           another_level_parameter="you get the idea"
#       }
#   }
#
# It tabs the whole level according to how many ":"s there
20 # are in the path name.
#
# Print_SDF_Level is one of a continuing series of recursive/
# iterative functions by Orion Pritchard. If Print_SDF_Level
# comes across another level (Conveniently called another_level
25 # in the example above), it calls Print_SDF_Level on that level.
#
# This function is used in:
#   -- internal to this file only--
#
30 #####
#
sub Print_SDF_Level
{
    my ($PARAMETER_ptr,$level,$suppress_comments) = @_;
35
    my @level_list = split (/\/,$level);
    my $number_of_tabs = (scalar (@level_list) - 1);
    my $spacing = (" " x (4*$number_of_tabs));
40
    my @leaves = (split (/\/,$$PARAMETER_ptr{"LEAF:$level"}));
    foreach $leaf (@leaves)
    {
        my $p = $$PARAMETER_ptr{"$level:$leaf"};
        my $e = $$PARAMETER_ptr{"EDIT:$level:$leaf"};
45 $e = "\[EDIT\]" unless $e;
        if ($p ne "")
        {
            my $string = "$spacing$leaf=\"$p\"\"";
            print "$string";
50 #print ((" "x(60-$string_length))."\# $e") if !$suppress_comments;
            print "\n";
        }
        else
        {
55 #print "$spacing$leaf\n";
            print "$spacing\{\n";
            &Print_SDF_Level ($PARAMETER_ptr,"$level:$leaf");
        }
    }
60 my $out_string = " " x (4*($number_of_tabs-1))."\}\n";
    print $out_string if ($number_of_tabs > 0);
}

```

```
#####
#
# Add_SDF_Node adds a node to the tree/parameter structure that
parse/print_sdf_files
5 # know so well. It adds the parameter value and also updates the LEAF
value(s).
# Add_SDF_Node is smart enough to add heirarchy if its needed.
# e.g.
# &Add_SDF_Node(\%PList,"a:b:c:d = foo") would ensure $$Plist{"a:b:c:d"} = foo
10 # and also make sure that $$Plist{"LEAF:a"} includes b, "LEAF:a:b" includes c,
etc.
#
# INPUTS
# $parameter_list: the "pointer" to the list that add_SDF_node should add to.
15 # $parameter: a string of the form "parameter = blerg".
# $options: "edit options", not required.
#
#
# This function is used in:
20 # mk_systembus.pl
# v2vpp.pl
#
#####
#
25 sub Add_SDF_Node
(
    my($parameter_list,$parameter,$options) = (@_);
    $options = "EDIT" if (!$options);
    $parameter =~ s/^\s*(.*)/$1/;
    $parameter =~ s/^(.*)\s*$/$1/;
    $options =~ s/^\s*(.*)/$1/;
    $options =~ s/^(.*)\s*$/$1/;

    #
    35 #param,$parameter,$options\n";

    #put the parameter in the heirarchy
    my ($parameter_lhs,$parameter_rhs) = split (/^\s*=\s*/, $parameter);

    40 # Don't let the user add null entries to the tree. It's just annoying:
    # return if $parameter_rhs eq "";

    $$parameter_list{$parameter_lhs} = $parameter_rhs;
    $$parameter_list{"EDIT:$parameter_lhs"} = $options;

    45 #put the leaf(ves) in the heirarchy. Only add
    #to leaf value if path has not been declared before.
    #Iterate over all levels in the path.

    50 my @path = split (/^\s*:\s*/, $parameter_lhs);
    while (scalar(@path) > 0)
    {
        #leaf_rhs is equal to last value in parameter_lhs
        my ($leaf_rhs) = pop(@path);
        55 my ($leaf_lhs) = join(':', @path);
        #print STDERR "in While loop, path is @path, l_rhs is $leaf_rhs, l_lhs is
        $leaf_lhs\n";

        my $leaf_string = $$parameter_list{"LEAF:$leaf_lhs"};
        60 if ($leaf_string !~ /\b$leaf_rhs\b/)
        {
            #print STDERR "leaf_string is $leaf_string, adding $leaf_rhs to
            melee\n";
        }
    }
}

```

```
next if $leaf_rhs eq "";
```

```

# Add a comma if the string exists, and it doesn't already end in a
comma.

```

```
#
$$parameter_list{"LEAF:$leaf_lhs"} .= ","
    if ( $$parameter_list{"LEAF:$leaf_lhs"} ne "" ) &&
        ( $$parameter_list{"LEAF:$leaf_lhs"} !~ /\,$/ ) ;
```

```

$$parameter_list{"LEAF:$leaf_lhs"} .= "$leaf_rhs";
$leaf_string = $$parameter_list{"LEAF:$leaf_lhs"};
#print STDERR "leaf_string now is $leaf_string\n";

```

```
#####
#
# Get_All_SDF_Parameters_At_RegExp_Level
# Recursively searches associative array %$pList for level indexes that match
# $RegExp. Starts at $Level, returns an array of a="b" if $Level
# matches
```

```
# $Reg_Expression (or just a if a="a").
# $Reg_Expression defaults to ./.* (match everything).
# This function preserves the order of the values in the ptf file
```

```
# This function is used in:
#   mk_systembus.pl
```

#####

```
sub Get_All_SDF_Parameters_At_RegExp_Level
```

```
my ($pList,$Level,$Reg_Expression) = @_;
my (@values);
```

```
$Reg_Expression = ".$*" if ($Reg_Expression eq "");
```

```
#warn "GASPAREL $Level\n  $Reg_Expression\n";
my $list = $$pList{"LEAF:$Level"};
foreach $entry (split (/\\s*,\\s*/, $list))
{
```

```
#warn "entry is $entry\n";
if ($Level =~ /$Reg_Expression/)
{
```

```
#warn "entry matched reg_exp\n";
my $entry_value = $$pList{"$Level:$entry"};
if ($entry eq $entry_value)
{
    push (@values, "$entry");
}
else
{
    push (@values, "$entry\\=$entry_value");
}
```

```

    }
    else
    {
        push (@values,&Get_All_SDF_Parameters_At_RegExp_Level($pList,
                                                                "$Level:$entry",
                                                                $Reg_Expression));
    }
}
return (@values);

```

```

}
#####
#
# Add_SDF_Parameters
5 #
# A simple subroutine to ease typing.  You pass in the module name and a string
# of the form "parameter1 =
$parameter1,parameter2=$parameter2,parameter3=$parameter3".
# If your parameters are global, you just need to pass the parameter name and
10 # this function takes care of the rest.
#
# Sometimes, the user might want to specify the parameter string
# on the Vpp command-line.  In this case, it's important that the
# parameter string not have any equals-signs in it.  To allow this,
15 # this routine interprets the magic token "__equals__" as an equals-sign.
#
# This function is used in:
# v2vpp.pl
#
20 #####
#

sub Add_SDF_Parameters
(
25 my ($Module_Name, $parameter_string, $section_name) = (@_);

$parameter_string =~ s/__equals__/=/g;
$section_name = "PARAMETERS" unless $section_name;

30 foreach $parameter (split(/\s*,\s*/, $parameter_string))
{
my ($lhs, $rhs) = split (/s*\s*=\s*/, $parameter);
$rhs = eval("\$".$lhs) if ($rhs eq "");
&Add_SDF_Node ($TL, ":MODULE $Module_Name:$section_name:$lhs=$rhs");
35 }
}

1; # Modules need to say "One!"
40

45

50

55

```

wiz_convert.pm

```
#####
# wiz_convert.pm
5 #
# Format-conversion subroutine library.
#
# So many formats, so little point.
#
10 #####

#####
# Convert_Srec_To_Mif
#
15 # Converts an S-record file (as might be emitted by
# the GnuPro compiler) into a mif-file suitable for initializing
# an APEX on-chip ROM.
#
# If you don't specify a base-address for the S-record
20 # contents, it just uses the first address it sees. You have
# been warned.
#
#####
sub Convert_Srec_To_Mif
25 {
    my ($srec_name, $mif_name, $srec_base_address) = (@_);
    open (SREC, "< $srec_name") or die "Couldn't open $srec_name: $!";
    open (MIF, "> $mif_name") or die "Couldn't open $mif_name: $!";

30     my $a;
    my $recordType;
    my $recordLength;
    my $recordAddress;
    my $recordData;
35     my $recordChecksum;
    my $residue;
    my $i;

    print MIF <<EOP;
40     WIDTH=16;
    DEPTH=512;
    ADDRESS_RADIX=DEC;
    DATA_RADIX=HEX;
    CONTENT BEGIN
45     EOP
        while($a = <SREC>)
        {
            if($a =~ /^S([123])(..)(.*)($/)# an S record we can use
50             {
                $recordType = $1;
                $recordLength = hex($2)-1;
                $a = $3;
                $recordChecksum = $4;

55                 $recordLength -= $recordType + 2;
                my $addressStringLength = ($recordType + 1) * 2;

                $recordAddress = hex(substr($a,0,$addressStringLength));
                $srec_base_address = $recordAddress if ($srec_base_address eq "");

60                 $recordAddress -= $srec_base_address;
                $recordData = substr($a,$addressStringLength);
```

09030106 "061201

09280106 "061201

```

        if($residue)
        {
            $recordAddress--;
            $recordData = $residue . $recordData;
5          }

        $recordLength = length($recordData) & ~3;
        $i = 0;
        while($i < $recordLength)
10        {
            printf MIF " %5d : %s%s;\n",$recordAddress/2,
                substr($recordData,$i+2,2),
                substr($recordData,$i,2);
            $recordAddress += 2;
15            $i += 4;
        }
        $residue = substr($recordData,$i);
    }
}

20 print MIF <<EOP;
END;
EOP

25 # We discovered this interesting fact about the S-Record file.
    # Perhaps our caller might like to know about it:
    #
    return $srec_base_address;
}

30 #####
# mif_num_to_decimal
#
# Given a value you got from a MIF-file, and the accompanying
35 # RADIX string (BIN, DEC, HEX, OCT, or UNS);
# This converts the number to base-10 and returns the result.
#
#####
40 my %mif_base = ( BIN => 2,
                  OCT => 8,
                  DEC => 10,
                  UNS => 10,
                  HEX => 16
                  );

45 my %mif_digits = (0=>0,1=>1,2=>2,3=>3,4=>4,5=>5,6=>6,7=>7,8=>8,9=>9,
                  a=>10,b=>11,c=>12,d=>13,e=>14,f=>15);

sub mif_num_to_decimal
50 {
    my($mif_string, $mif_radix) = (@_);

    my $base = $mif_base{$mif_radix};

55    $mif_string = lc($mif_string);

    my @char_list = split ("", $mif_string);
    my $result = 0;
    foreach $char (@char_list) {
60        $result *= $base;
        $result += $mif_digits{$char};
    }
    return $result;
}
```

```

}

#####
# Emit_Bin_Data
#
5 # Just a utility routine, used only by Convert_Mif_To_Dat,
# below.
#
# Presumes the global filehandle DAT is open for writing.
10 #
#####
sub Emit_Bin_Data
{
15     my ($data_val, $data_width) = (@_);

    my $bit = 0;
    my $result = "";
    for ($bit = 0; $bit < $data_width; $bit++)
    {
20         if ($data_val % 2 == 0) {
            $result = "0" . $result;
        } else {
            $result = "1" . $result;
        }
25         $data_val = int ($data_val / 2);
    }

    print DAT "$result\n";
30 }

#####
# Convert_Mif_To_Dat
#
35 # Verilog simulations allow you to initialize memory with
# .dat-files. Quartus allows you to initialize memory with
# .mif-files. One could write a Perl-script to convert
# mif-files into dat-files. Oh, look! One did!
#
40 # This function takes, as arguments, the name of the mif-file,
# the name of the dat-file, and the width of the memory-to-be-
# initialized, in bits.
#
45 #####
sub Convert_Mif_To_Dat
{
    my ($mif_name, $dat_name) = (@_);

50     # Load mif-lines into a big string
    open (MIF, "< $mif_name") or die "Couldn't open $mif_name: $!";
    my $mif_contents = "";
    while (<MIF>)
    {
55         # While we've still got $_ as a line, strip-off "--" comments:
        s/^(.*?)--.*$/\1/;
        $mif_contents .= $_;
    }
    close (MIF);

60     # Eliminate /* - */ comments:
    while ($mif_contents =~ s/(.*?)\\\/\*.*?\*\/(.*?)\1 $2/sg) {};

```

Eliminate loathsome whitespace:

\$mif_contents =~ s/\s+//sg;

my \$mif_addr_radix = "HEX"; # Default, as-shown in mif-spec.

my \$mif_data_radix = "HEX";

my \$mif_width = "";

I don't happen to care about the depth.

\$mif_addr_radix = \$1 if \$mif_contents =~ /ADDRESS_RADIX=(\w+)/is;

\$mif_data_radix = \$1 if \$mif_contents =~ /DATA_RADIX=(\w+)/is;

\$mif_width = \$1 if \$mif_contents =~ /WIDTH=(\w+)/is;

\$mif_depth = \$1 if \$mif_contents =~ /DEPTH=(\w+)/is;

if ((\$mif_width eq "")) {

printf STDERR "Convert_Mif_To_Dat: suspicious mif-file: \$mif_name\n";

printf STDERR " no WIDTH specified.\n";

return -1;

}

if ((\$mif_depth eq "")) {

printf STDERR "Convert_Mif_To_Dat: suspicious mif-file: \$mif_name\n";

printf STDERR " no DEPTH specified.\n";

return -1;

}

open (DAT, "> \$dat_name") or die "Couldn't open \$dat_name: \$!";

if (\$mif_contents !~ /CONTENTBEGIN(.*)END\;/is) {

warn "Convert_Mif_To_Dat: no contents in mif file: \$mif_name";

return -1;

}

my \$contents_section = \$1;

my @statement_list = split (/\/, \$contents_section);

foreach \$statement (@statement_list) {

next unless \$statement =~ /^([\dABCDEF]+\):([\dABCDEF]+)\$/i;

my \$addr = \$1;

my \$data = \$2;

my \$dec_addr = &mif_num_to_decimal (\$addr, \$mif_addr_radix);

my \$dec_data = &mif_num_to_decimal (\$data, \$mif_data_radix);

printf DAT ("%X\n", \$dec_addr);

&Emit_Bin_Data(\$dec_data, \$mif_width);

}

close (DAT);

}

For some reason, modules need a return-value.

here's mine:

"Abraham Lincoln";

09630106-061201

<u>Docket No.</u>	<u>Notes</u>	<u>Due Dat</u>	<u>Initial</u>
KLA1P012.WO	design. EP, JP with ISA-US	07-Jun	T
VISAP015.WO	nat'l in AU, CA and EP	08-Jun	H
SUN1P270.WO	all (except US) with EP/ISA	09-Jun	H
SUN1P800.WO	all (except US) with EP/ISA	12-Jun	H
KLA1P004.WO	nat'l in EP and JP	17-Jun	H
SAY1T001	trademark filing in EP, JP, CA and AU	20-Jun	H
CYTOP003.WO	all (including US)	23-Jun	H
LAM1P098.WO	nat'l in EP (FR,DE,GB,IT) JP,KR,SG	28-Jun	T
LAM1P097.WO	nat'l in EP(BE,FR,DE,GB,IT,NL,ES) IL,JP,KR,SG	29-Jun	K
LAM1P139.WO	all including the US with EP/ISA and TW	30-Jun	T
LAM1P140.WO	all including the US with EP/ISA and TW	30-Jun	K
LAM1P141.WO	all including the US with EP/ISA	30-Jun	T
SUN1P264.WO	all (except US) with EP/ISA	30-Jun	
CAMIP002.WO	all (except US) with EP/ISA	10-Jul	
CAMIP004.WO	all (except US) with EP/ISA	10-Jul	
IMECP006.WO	instructions to follow	11-Jul	K
IMECP007.WO	instructions to follow	11-Jul	H
IMECP008.WO	instructions to follow	11-Jul	T
IMECP009.WO	instructions to follow	11-Jul	K
SUN1P399.EP	nat'l filing in EP (DE,FR,GB) (file in July)	25-Jul	T
SCHIP002.WO	nat'l filing in CN,JP,KR,EP (DE,FR,IE,IT,GB)	28-Jul	K
SCHIP022.WO	nat'l filing in CN,JP,KR,EP (DE,FR,IE,IT,GB)	28-Jul	H
CAMIT003/A	trademark filing in EP, JP, CA and AU	02-Aug	K
CAMIT002/A	trademark filing in EP, JP, CA and AU	02-Aug	K
SCHIP003.WO	nat'l filing in CN,JP,KR,EP (DE,FR,IE,IT,GB)	22-Aug	K
SUN1P707.EP	nat'l filing in EP (DE,FR,GB) (file in August)	31-Aug	H
AVI1P003	nat'l filings in DE and JP	08-Sep	K
TRIPP030PX1.WO	further instructions to follow - due 2/28/02		

wiz_utils.pm

```
#####
# wiz_utils.pm
#
# A set of utility routines and global variables
# which are useful for automating the wiard "back-end"
# build proecss. These include:
#
# * Path names for all the Nios HDK source directories.
#
# * A handy subroutine for parsing named function arguments.
#
# * A handy subroutine for stripping Perl comments.
#
# * Routines for copying files and directories platform-independently.
#

# We would like to auto-flush our buffers in the new, fancy
# way (using ->autoflush()), but for now we'll do it in the
# old, cryptic way (using $|), because we can't seem to find
# our library modules.
#use IO::Handle;
$| = 1;      # set flushing on STDOUT
my $wiz_util_old_fh = select (STDERR);
$| = 1;      # set flushing on STDERR
select ($wiz_util_old_fh);

use vpp;
use ptf_update;
use mk_custom_sdk;    # Just to get formatted-print function "print_command."
                      # --Inefficient.  Sould put in some shared
                      # module.

#####
# Progress
#
# Prints-out a happy progress message in a standard format.
#
#####
sub Progress
{
    foreach $msg (@_)
    { &print_command ($msg) }    # Use DvB's unimprovable format.
}

$LEONARDO_EXEC=$ENV{"SPECTRUM_ROOTDIR"};
$LEONARDO_EXEC=~ s/\\/\\/g;    # I hate '\'.

$PERL_PROGRAM_FILE = $ENV{"JPERL_PERL_CMD"};
$PERL_PROGRAM_FILE = "perl" if !defined ($PERL_PROGRAM_FILE);

#####
# Build_Cmd_Line
#
# Turning a bunch of strings into a command-line isn't as easy
# as one might think.  First of all, it's safest to double-quote
# every argument--especially since filenames might very well have
# spaces in them.
```

```

#
# Second, clients may have already preemptively double-quoted
# the arguments they're passing-around, so I want to be careful
# to not put -another- layer of quotes on an argument if it's
5 # already been quoted.
#
# and, finally, we want to do all this in a way that works
# on 98/NT/2000/Solaris/HPUX. Whee.
#
10 # This function takes an arbitrarily-long list of strings,
# each of which is taken as a single argument for a shell-command.
# This function returns a single command-line (string), which is
# the union of all the arguments, separated by spaces. Each argument
# may contain spaces. Each argument in the resultant command line
15 # will be double-quoted.
#
# A single element in the list --may-- consist of multiple command-line
# arguments. If so, each one must be double-quoted, and they must
# all be separated by spaces.
20 #
# This function also handles I/O redirection properly, and doesn't
# molest (quote) I/O redirection specifiers (>,<,&,&) if they appear in
# an argument all by themselves.
#
25 # ---> Significant consideration:
#
# On Windows platforms, the "system" command WILL NOT WORK if the
# command-name (0th arg) is double-quoted. This is odd, since it
# works fine at the interactive command line. Nevertheless, bitter
30 # experience shows that perl's "system" feature will not work for
# double-quoted program names. So we don't double-quote the program
# name. This means that "Build_Cmd_Line" won't work for
# program names with spaces in them. For the most part, for the
# Nios HDK, this means (ahem):
35 #
# PLEASE DO NOT INSTALL QUARTUS UNDER A PATH WITH SPACES IN IT.
#
# Thank you.
#
40 #####
sub Build_Cmd_Line
{
    my @cmd_args;
    (@cmd_args) = (@_);
45
    my @broken_args; # Break-apart any already-double-quoted commands.

    foreach $argo (@cmd_args)
    {
50         # No quotes in argument: Just pass it along.
        push (@broken_args, $argo), next if $argo !~ /\\"/;

        # There has to be a double-quote at both ends of the
        # argument, or else it's not valid. Strip-off leading/trailing
55         # quotes to make splitting easier:
        #
        $argo =~ s/^\"s*\"(.*)\"s*\"/$1/ or die
            "Build_Cmd_Line: Illegal use of double-quotes:
60             [$argo]";

        my @sub_args = split (/\\"s+\"/, $argo);
        foreach $sub_arg (@sub_args)
            { push (@broken_args, $sub_arg) }
    }
}

```

```

}

$result = "";
foreach $argo (@broken_args)
5 {
    # Choke if there's a double-quote buried in the argument:
    $argo !~ /[^\\"\\\"/ or die
        " ERROR: Build_Cmd_Line: argument contains double-quote:
          [$argo].";

10     if ($result eq "")
        {
            # First argument is special. We don't quote it, and it
            # can't have spaces. (See rant, above).

15             $argo !~ /\s/ or die
                "It seems that you have installed Quartus in a
                  directory-path which has spaces in its name:

20                 $argo

                    The Nios HDK simply will not work under these
                    conditions. It is very regrettable, and extremely
                    sad. Yet, alas, it is also true.";

25             $result = $argo . " ";
        } else {
            # Not the first argument.
            # Quote it if it's not an I/O redirection thingamabob:
            $argo =~ s/\s*(.*?)\s*/$1/; # strip leading/trailing spaces.

30             $argo = "\" . $argo . "\"" if $argo =~ /[^\>\<\\|&]/;

            $result .= $argo . " ";
        }
    }
    return $result;
}

40 #####
# Run_System_Command
#
# THE FIRST ELEMENT OF THE ARGUMENT LIST IS AN ERROR MESSAGE!
#
45 # You pass a -list- of arguments (some of which may contain
# spaces). This function builds them into a properly-formed
# command line, runs the command-line via "system," checks
# for an error, and prints an informative error message if it happens.
#
50 # If the system command results in an error, this function
# does not return.
#
#####
sub Run_System_Command
55 {
    my $error_msg;
    my @cmd_args;
    ($error_msg, @cmd_args) = (@_);

60    my $cmd_line = &Build_Cmd_Line (@cmd_args);

    my $num_words = scalar (@cmd_list);

```

09380106 "061201
TOTAL 90

```

# comment-out these flusharoos until we can make IO:Hanlde work.
#STDOUT->autoflush(1);
#STDERR->autoflush(1);

5  my $run_banner =<<END_RUN_BANNER;
    *****
    * About to Run System Command:
      $cmd_line
    *****

10  END_RUN_BANNER

    print STDOUT $run_banner;
    print STDERR " ";

15  system ($cmd_line) == 0 or die
      "ERROR ($?) running system command line.
        *** $error_msg
          [$cmd_line] ";

20  }

#####
# Log
#
25 # Logs messages to either one, or both, of two log-files
# at the same time.
#
# One log-file is a high-level "scorecard." This file should
# contain progress messages for "very big" operations only.
30 # When your giant 72-hour regression test is complete, A
# user should be able to print-out this file on a few pages
# and determine, from 10,000 feet, what happend and how it went.
#
# The second log file is lower-level progress report. It should contain more
# fine-grained progress messages, but still nothing like
35 # the aggregated STOUT and STDERR, which will be megabytes and
# megabytes of crap.
#
# The first argument to this funciton is logged (with a timestamp)
40 # to the scorecard file. Null strings ("") are not logged.
# The first argument is also logged to the progress report file.
#
# The second argument is logged (with a timestamp) to the
# progress report. Null strings ("") are not logged.
45 #
# **** Where are the log files?
#
# By default, the scorecard log messages will go to a file named
# "./scorecard.log", and the progress report messages will go to a file
50 # named "./progress_report.log"
#
# (The messages are also emitted to STDOUT and STDERR).
#
# There is not currently any way to change the default behavior.
55 # I might do this someday if I have time.
#
#####
sub Log_Guts
(
60   my $log_file_name;
   my $msg;
   my $do_timestamp;
   my $do_send_to_stdout;

```



```
( $log_file_name, $msg, $do_timestamp, $do_send_to_stdout) = (@_);
```

```
return if !defined ($msg);
```

```
return if $msg eq "";
```

```
my $time_string = localtime();
```

```
my $uber_msg = "$msg\n";
```

```
$uber_msg = "--> $time_string <--\n$uber_msg" if $do_timestamp;
```

```
open (LOG_FILE, ">> $log_file_name") or die
```

```
"Log: Couldn't open $log_file_name: $!";
```

```
print LOG_FILE $uber_msg;
```

```
close (LOG_FILE);
```

```
if ($do_send_to_stdout) {
```

```
    print STDOUT $uber_msg;
```

```
}
```

```
}
```

```
sub Log
```

```
{
```

```
    my $score_msg;
```

```
    my $prog_msg;
```

```
    ($score_msg, $prog_msg) = (@_);
```

```
    my $scorecard_file = $ENV{NIOS_SCORECARD};
```

```
    my $progress_file = $ENV{NIOS_PROGRESS_REPORT};
```

```
    $scorecard_file = "./score_card.log" if $scorecard_file eq "";
```

```
    $progress_file = "./progress_report.log" if $progress_file eq "";
```

```
    my $progress_only = $score_msg eq "";
```

```
    &Log_Guts ($scorecard_file, $score_msg, 1, 1);
```

```
    &Log_Guts ($progress_file, $score_msg, 1, 0);
```

```
    &Log_Guts ($progress_file, $prog_msg, 0, $progress_only);
```

```
}
```

```
#####
```

```
# Find_SOPC_Component_Directory
```

```
#
```

```
# In an ideal world, a user would be able to take a system's PTF-file,
```

```
# copy it to another computer (or directory), and reproduce the
```

```
# original system from it--even if the new computer has the
```

```
# SOPC-Builder library installed in a totally different place.
```

```
#
```

```
# To accomplish this, we need to do "run-time binding" of the SOPC-Builder
```

```
# library/libraries. This means we have to go out and find them
```

```
# when the user actually runs the tool, based on some kind of
```

```
# search-path.
```

```
#
```

```
# The search-path is handed to us by the caller as one of those
```

```
# dash-dash command-line arguments. If no search path is specified,
```

```
# we use a sensible default (set up in "Process_Wizard_Script_Arguments,"
```

```
# below).
```

```
#
```

```
# This function uses the library-search path to find the correct
```

```
# directory containing the given component-class name.
```

```
#
```

```
# It does so by going through all of the directories in the path and
```

```
# looking for a -subdirectory- which contains a "class.ptf" file. If
```

```
# we find one, we open it and see what "class" it is. We return the
```

```
# -subdirectory name- in which we found the matching "class.ptf" file.
```

09330106 061201

```

#
#####
sub Find_SOPC_Component_Directory
(
5   my ($module_class, $path_string) = (@_);

   # Path-elements can be split by numerous oddball characters.
   # this might very well come in handy later:
   #
10  my @path_dirs = split (/\\s*[\.,\;\|\+=\!\&\^\#\]\s*/, $path_string);

   my $result = "";   # Subdirectory in which we found matching "class.ptf"

   foreach $dir (@path_dirs) {
15     # It's polite to tolerate directory-names both with- and without
     # trailing slashes.  If we see a trailing slash, we get rid of it:
     # (and we convert all evil backslashes into good forward-slashes.
     $dir =~ s|\\|\/|g;
     $dir =~ s|\/$||g;

20     # Get a list of all subdirectories.  Actually, we just get a list
     # of all the files, and then ignore them if they're not
     # directories.
     #
25     if (!opendir (DIR, $dir)) {
       warn ("Couldn't open SOPC library directory '$dir' : $!");
       next;
     }
     my @subdirectories = (readdir(DIR));
30     closedir (DIR);

     foreach $sub_dir (@subdirectories) {
       my $sub_dir_path = "$dir/$sub_dir";
       my $class_ptf_filename = "$sub_dir_path/class.ptf";

35       next if !-d $sub_dir_path;           # Must be a directory...
       next if !-e "$class_ptf_filename";   # with a class.ptf file.

       my $db_PTF_file = new_ptf_from_file ($class_ptf_filename);
40       next if !$db_PTF_file;               # If we can't open it: forget it.

       my $db_Class = &get_child_by_path ($db_PTF_file, "CLASS");
       next if !$db_Class;                   # No 'class' section: forget it.

45       my $found_class_name = &get_data ($db_Class);
       next if $found_class_name ne $module_class;

       # If we got to here, then we've found a "class.ptf" file which
       # defines the class we're looking for.  I suppose I could
50       # do other checks, like look for a valid "MODULE_DEFAULTS" section,
       # but that's a slippery slope.  For now, a "class.ptf" file which
       # defines the class we're looking for is good enough.
       #
       $result = $sub_dir_path;
55       last;
     }
     last if $result;
  }

60  if (!$result) {
    warn ("
      SOPC-Builder library component '$module_class' not found.
      Could not find a 'class.ptf' file which defines '$module_class'

```

```

        on the path: ($path_string)\n");
    }

```

```

    return $result;
}

```

```

#####

```

```

# Process_Wizard_Script_Arguments

```

```

#
# At the top of each X-wizard script (mk_<X>.pl) there was
# a little preamble of code which analyzed the PTF file and
# set some global variables in a highly-ritualized way.
#

```

```

# Did someone say "highly ritualized?" This sounds like a job
# for a subroutine. I'm so lazy. That's why I'm a good Perl
# programmer.
#

```

```

# This is just a wrapper around &Parse_Named_Arguments which,
# additionally:
#

```

```

# * Reads the "arguments" to this script out of the (indicated) PTF file,
# and re-phrases them as an argument list which is interpreted
# by the aforementioned &Parse_Named_Arguments.
#

```

```

# * Sets the global variable $QUARTUS_PROJECT_DIR
#

```

```

# * Sets the global variable $MODELSIM_DIR
#

```

```

# * Returns a string of all the input arguments with the
# equals-signs substituted with "__equals__".
# (useful for passing arguments down to PTF-file).
#

```

```

# * "Decodes" spaces in input-arguments.
# we've found that it's difficult to pass
# script-arguments around if they contain whitespace.
# solution: Replace " " with "__jperl_space__". This
# function does the reverse, acting as a receiver for
# this sneaky encoding.
#

```

```

# This routine has one (and maybe later more) arg which is
# always accepted: "wizard." This is saved-away in the PTF-file
# and used by the wizards themselves later to figure out
# who ran this function (which wizard), and who should be called
# when this device needs to be edited.
#

```

```

# The use of the term "arguments" in this function is a bit tricky.
# There are really two distinct sets of things we call "arguments":
#

```

```

# 1) The actual arguments passed to this here function, which
# specify the current working directory and PTF file and stuff.
#

```

```

# 2) The contents of the WIZARD_SCRIPT_ARGUMENTS" section of the
# PTF file
#

```

```

# We have to process the type (1) arguments in order to -get at- the
# type (2) arguments. The result of this function, a hash, is obtained
# by opening the PTF file (based on the (1) arguments) and then recasting
# the contents of the PTF-file (the type (2) arguments). How very
# confusing. Sorry. That's what happens when there are multiple levels
# of indirection.
#

```

```

$Process_Wiz_Args_Doc=<<<END_OF_DOCUMENTATION_STRING ;

```

09330105-051201

[hippie] --tolerate unfamiliar arguments.

```
5  # LONG NAME          SHORT   DEFAULT   DESCRIPTION
# -----
* system_directory    --none--   .         Directory where system resides.
* target_module_name   name      --none--   Module being generated.
* system_name         --none--   --none--   Name of system being generated.
* socp_directory      --none--   --none--   Where the SOPC-Bldr is installed.
10 * socp_lib_path       --none--   --none--   Where to look for lib dirs.
    generate          --none--   1         "Yes, please do generate, please."
    verbose           v          0         *bool* Extra blabbering output.

# Just for sheer convenience, we also provide the client (caller) with
15 # "fictitious" arguments called:
#
* class_directory     --none--   --none--   Where THIS component lib dir is.
    system_sim_dir     sim_dir  --none--   If sim project, where to put it.

20 END_OF_DOCUMENTATION_STRING
#
#####
sub Process_Wizard_Script_Arguments
{
25     my ($arg_doc_string, @input_arg_list) = (@_);

    #####
    # Pre-process argument string.
    #
30     # I thought I'd get a list of space-delimited arguments.
    # Instead, I just get one single argument, which is a big ol' string.
    # I'll just write a little bit of code which works either way:
    my @intermediate_arg_list = ();
    my @quoted_string_list = ();
    foreach $in_arg (@input_arg_list)
    {
35         while ($in_arg =~ s/\\"([^\"]*)\\"/\\"__ARG_QUOTED_STRING__\\"/)
        {
            push (@quoted_string_list, $1);
40         }

        push (@intermediate_arg_list, split (/\\s+/, $in_arg));
    }

45     foreach $arg (@intermediate_arg_list)
    {
        while ($arg =~ /\\"__ARG_QUOTED_STRING__\\"/)
        {
50             my $next_quoted_string = shift (@quoted_string_list);
            $arg =~ s/\\"__ARG_QUOTED_STRING__\\"/$next_quoted_string/;
        }
        push (@processed_arg_list, $arg);
    }

55     #####
    # First, we expect to see a certain, predefined set of
    # incoming arguments.
    #
    # Note that these incoming arguments are re-interpreted as part of
60     # our fictitious name=value "argument" list, as well as processed
    # directly here.
    #
    my @name_equals_value_list = ();
```

09880106_061201
T02T90" 90T08860

```

my $sys_dir      = "";
my $sys_name     = "";
my $mod_name     = "";
my $lib_path     = "";

5
while ($arg = shift (@processed_arg_list))
{
    # We expect arguments to be name=value pairs that begin
    # with double-dashes.  Hmm.
10    $arg =~ s/^--// or die "malformed argument: $arg";
    $arg =~ /[^(=)]+=[^(=)]*/ or die "malformed argument: $arg";
    push (@name_equals_value_list, $arg);
    my $arg_name = $1;
    my $arg_value = $2;

15    $sys_dir = $arg_value if ($arg_name =~ /^system_directory/);
    $sys_name = $arg_value if ($arg_name =~ /^system_name/);
    $mod_name = $arg_value if ($arg_name =~ /^target_module_name/);
    $sopc_dir = $arg_value if ($arg_name =~ /^sopc_directory/);
20    $lib_path = $arg_value if ($arg_name =~ /^sopc_lib_path/);
    $verbose = $arg_value if ($arg_name =~ /^verbose/);
}

# Library path default:
#   If no library path was specified, we use a sensible default:
#
25 if ($lib_path eq "") {
    $lib_path = "$sopc_dir/components";
    push (@name_equals_value_list, "sopc_lib_path=$lib_path");
30 }

# This is a bit bogus, but it's a thing several people might need
# to know, and which we can figure out here.
push (@name_equals_value_list, "system_sim_dir=$sys_dir/$sys_name\_sim");
35

&Progress ("Extracting PTF info for $mod_name.") if $verbose;

$msg = "Couldn't process PTF-file arguments for module $mod_name.";
40 my $ptf_filename = "$sys_dir/$sys_name.ptf";

&PTF_Translate_Old_Version ($ptf_filename); # Update legacy files.

my $db_PTF_File = &PTF_New_Required_Ptf_From_File ($ptf_filename, $msg);
45 my $db_Sys = &PTF_Get_Required_Child_By_Path ($db_PTF_File,
                                                "SYSTEM", $msg);

#####
# The "-target_module_name" argument is special.  If it has the
# exact-same name as the system itself, then we get the
50 # WIZARD_SCRIPT_ARGUMENTS from the SYSTEM section itself, instead
# of one of its sub-modules.
#
my $db_Module = $db_Sys;
$db_Module = &PTF_Get_Required_Child_By_Path
55 ($db_Sys, "MODULE $mod_name", "That's odd.")
    unless $mod_name eq $sys_name;

my $db_Wiz_Args = &get_child_by_path ($db_Module,
                                      "WIZARD_SCRIPT_ARGUMENTS");
60

#####
# The fictitious "class_directory" argument
#

```

09880106 "061201

```

# We -wish- the user had passed us yet-another command-line argument
# called "--class_directory" The user did not because it is, after
# all, something we could figure out for ourselves. Please allow
# me to demonstrate:
#
5 if ($mod_name eq $sys_name)
{
    # If We're generating the system (and bus) itself, then
    # the "class_directory" is just the directory in which the
10    # system-builder library stuff lives:
    #
    # NOTE: JWIZ NAME CHANGE
    # The name of this directory should be changed when we
    # re-name all the components. I think. Maybe.
15    push (@name_equals_value_list,
        "class_directory=$sopc_dir/bin");
} else {
    # This is just an ordinary module, so we search for its
    # class-directory in the conventional manner:
20    #
    my $class = &PTF_Get_Required_Data_By_Path ($db_Module, "class");
    my $class_dir = &Find_SOPC_Component_Directory ($class, $lib_path);
    push (@name_equals_value_list, "class_directory=$class_dir");
}
25 #####
# For compatibility with the old-style (and very powerful)
# &Parse_Named_Arguments function, we convert all the assignments
# in the fetched WIZARD_SCRIPT_ARGUMENTS section into
30 # a Perl-list of "name=value" strings. This lets us use all our
# old, familiar argument-parsing-and-checking infrastructure. How
# civilized.
#
my $num_wiz_args = &get_child_count ($db_Wiz_Args);
35 for ($child_index = 0; $child_index < $num_wiz_args; $child_index++)
{
    my $db_Arg = &get_child ($db_Wiz_Args, $child_index);
    push (@name_equals_value_list,
        &get_name ($db_Arg) . "=" . &get_data ($db_Arg)
40        );
}

# Take spaces from Perl as a special token. This helps us smuggle
# them past the command line.
45 #
# This is almost totally anachronistic, but c'est la vie, eh?
# There's no reasyn really to quit doing this.
#
50 my $named_arg_string = join (" ", @name_equals_value_list);
    $named_arg_string =~ s/\n/ /mg; # form a single line, please.
    $named_arg_string =~ s/_jperl_space_/ /g;

55 my ($arg, $user_defined, $table) =
    &Parse_Named_Arguments (" $arg_doc_string \n $Process_Wiz_Args_Doc",
        $named_arg_string);

60 # DELETE ME:
# This code is obsolete. Anyone relying on these variables
# is in grave danger.
#
$QUARTUS_PROJECT_DIR = $$arg("system_directory") or die

```

09330106 "061201

"ERROR: No Quartus project directory specified.";

\$QUARTUS_PROJECT_DIR =~ s/\\/\\/g; #Crush backslashes in project dir.
\$QUARTUS_PROJECT_DIR =~ s/\/\$//; # strip trailing "/" from wd.

\$MODELSIM_DIR = "\$QUARTUS_PROJECT_DIR/\$PROJECT_SIM_SUBDIR_NAME";

return (\$arg, \$user_defined, \$db_Module, \$db_PTF_File);

}

Perlcopy
#

"cp" is different on every platform. Worse, sometimes you
get read-only files as a result, which is never what we want
(in this particular application).
#

One way to copy files is via a Perl-routine. This
routine opens the source-file, reads it into a list of lines,
closes it, opens the destination file, writes the list of lines,
then closes it.
#

As an added bonus, you may pass-in an optional regexp which gets
applied to every line on its way through. Most people will
never use this, but it sometimes comes in handy.
#

Crude, but effective.
#

sub Perlcopy

{
my (\$src, \$dest, \$regexp) = (@_);

\$src =~ /.*?([^\\\/]+)\$/;
my \$src_root = \$1;

If the destination is given as a directory, add-on the
root filename.

\$dest .= \$src_root if \$dest =~ /[\\\/]\$/;

my @lines;

open (SRC, "< \$src") or die
"Perlcopy: cannot open source file \$src: \$!";
while (<SRC) { push (@lines, \$_) }
close (SRC);

open (DST, "> \$dest") or die
"Perlcopy: cannot open destination file \$dest: \$!";

foreach \$line (@lines) {
if (\$regexp) {
eval ("\"\$line =~ \$regexp");
die "Perlcopy error (\$?) evaluating expression: \$regexp" if \$@;
}
print DST \$line;
}
close (DST);

CopyDir

```

#
# Copies all files in the $src directory to the $dest directory.
#
#####
5  sub CopyDir
  {
    my $src;
    my $dest;
    ($src, $dest) = (@_);

10    $dest =~ /[\\\/]$/ or die
        "ERROR: CopyDir destination ($dest) must be a directory.";

    opendir (DIR, $src) or die
15    "ERROR: CopyDir can't open directory $src: $!";

    my $fname;
    while (defined($fname = readdir(DIR)))
    {
20        next if $fname =~ /\^\.+$/;
        &Perlcopy ("src/$fname", $dest) ;
    }

    closedir (DIR);
25  }

#####
# Create_Dir_If_Needed
#
# The name pretty much says it all, don't it?
#
#####
30  sub Create_Dir_If_Needed
  {
35    my $DIR_NAME;
    ($DIR_NAME) = (@_);

    if (!-e $DIR_NAME)
    {
40        # 511 is octal '777'. No one here can
        # figure out how to make an octal number.
        #
        mkdir ($DIR_NAME, 511) or die
            "Error creating dir $DIR_NAME: $!";
45    }
  }

#####
# Copy_Tool_Control_Files
#
50  # Some of the more complex wizard-generated modules have,
# for example, "black boxes." Others have RAM or ROM components.
# All of these things are rather tricky to support.
#
55  # A scheme has been devised for synthesizing, compiling, and
# simulating "complex" components containing black-boxes and
# such. The details of this scheme are beyond the scope of this
# comment. Suffice it to say that the scheme requires the presence
# of a variety of "magic" files in the project directory.
60  #
# This function copies all the "magic" files needed to
# support complex components. One-stop shopping. For some
# not-so-complex components, you may end up with more files than you

```



```

# really needed. That wouldn't be the end of the world.
#
# You pass-in (a reference to) the %arg-hash used by mk_systembus.
# That contains special entries telling us where all the various
5 # directories are..
#
#####
sub Copy_Tool_Control_Files
{
10     my ($arg) = (@_);

    #####
    # If you aren't simulating, you really only need one file:
    # good ol' "leonardo_define.v"
15     #
    # If you are simulating, then we do some extra work to set up
    # the simulation directory.
    #
    # We used to use Perlcopy to move a flat-text version of leonardo_define.v
20 # into project directory. Unfortunately, now we need to explicitly spit it
    # out, so that it may be encrypted, if need be.

    # &Perlcopy ("$$arg{sopc_directory}/bin/vpp/leonardo_define.v",
    #           "$$arg{system_directory}/");

25     open (ALTERA_FILEHANDLE, "> $$arg{system_directory}/leonardo_define.v")
        or die "Unable to open $$arg{system_directory}/leonardo_define.v";
    print ALTERA_FILEHANDLE "$GLOBAL_COPYRIGHT_NOTICE\n";
    print ALTERA_FILEHANDLE "`define LEONARDO_SPECTRUM true";
30     close ALTERA_FILEHANDLE;

    return unless $$arg{do_build_sim}; #----> Adios, muchachos! <----

    &Debug (0, "Copying simulator files to $$arg{system_sim_dir}");

35     #####
    # Build simulation directory, copy-in a few special files:
    #
    &Create_Dir_If_Needed ($$arg{system_sim_dir});
    &Create_Dir_If_Needed ("$$arg{system_sim_dir}/work");

40     &Perlcopy
        ("$$arg{sopc_directory}/bin/vpp/modelsim_define.v",
        "$$arg{system_sim_dir}/");

45     &Perlcopy
        ("$$arg{sopc_directory}/bin/modelsim_support/test_equipment.v",
        "$$arg{system_sim_dir}/");

50     &Perlcopy
        ("$$arg{sopc_directory}/bin/modelsim_support/_info",
        "$$arg{system_sim_dir}/work/");

    # Copy over the magic "sim.mpf" file, with one substitution:
55     my $test_bench_name = "$$arg{system_name}_test_bench";
    &Perlcopy
        ("$$arg{sopc_directory}/bin/modelsim_support/sim.mpf",
        "$$arg{system_sim_dir}/$test_bench_name.mpf",
        "s/SOPC_TEST_BENCH_NAME/$test_bench_name/g"
60     );
}

#####

```

09330106 "06.1.201

```

# Firm_Flip_Flop Renaming...
#
# We have to create a bunch of custom variants of the file
# "Firm_Flip_Flop.v" The Nios core uses three variants:
5 #
#     Address_Out_Reg
#     Control_Out_Reg
#     Data_In_Reg
#
10 # The system-bus module creates yet more.
#
# All our "firm flip flop" WISIWYG register modules get the same
# processor-name prefix as all the rest of the Verilog files.
# We use the port list from "Firm_Flip_Flop" to build-up port lists
15 # for its (identical) variants. The reason we need these variants
# at-all is because they will end up with different .esf-files. That
# all gets handled at the PBM-level, though. The Nios core itself
# can't know whether or not these signals go off-chip.
#
20 # The firm flip-flop file gets created via a call to Vpp.
#
# THIS SUBROUTINE CALLS &Vpp.
#
# DO NOT CALL THIS SUBROUTINE FROM WITHIN &Vpp, because
25 # &Vpp is not reentrant.
#
#####
sub Create_Firm_Flip_Flop_Variant
{
30     my ($system_directory,
        $sopc_directory,
        $part_type,
        @variant_names) = (@_);

35     die "ERROR Create_Firm_Flip_Flop_Variant: PART_TYPE SETTING ($part_type)
        NOT SPECIFIED"
        unless ($part_type);

    foreach $variant_name (@variant_names)
    {
40         &Vpp ("-Q",
            "-O", "$system_directory/$variant_name.v",
            "FIRM_FLIP_FLOP_MODULE_NAME = $variant_name",
            "PART_TYPE = $part_type",
45             # NOTE: HARD DEPENDENCY ON NIOS COMPONENT
            #     It would be a good idea, in the future, for
            #     this function to be un-linked from the exact
            #     location of the nios-component library directory.
            #     Make no mistake: It's done this way because of
50             #     expediency and safety. Many an evil has been
            #     committed in the name of meeting the ship-date,
            #     and this is a case-in-point. It's defensible since
            #     this is, after all, the Nios kit. If there's no
            #     nios-component, it's not much of a kit.

55             "$sopc_directory/components/altera_nios/vpp_source/firm_flip_flop.vpp",
            );
        }
    }

60 #####
# Create_ESF_File
#

```

09880106 "061201
T.02190"

```
# Creates an ESF-file which contains the text you specify as
# its body.
#
# You can create multiple ESF-files at the same time, all with the
5 # same body, by passing-in a list of filename(roots).
#
#####
sub Create_ESF_File
{
10   my ($esf_body, $system_directory, @root_filename_list) = (@_);

   foreach $esf_filename_root (@root_filename_list) {
       my $esf_name = "$system_directory/$esf_filename_root.esf";

15       open (ESF_FILE, "> $esf_name") or die "couldn't open $esf_name: $!";

       print (ESF_FILE "\n
           OPTIONS_FOR_INDIVIDUAL_NODES_ONLY
20           {
               $esf_body
           }
           ");

       close (ESF_FILE);
25   }
}

#####
# Get_Direction_From_Avalon_Type
#
# Given one of those crazy Avalon port-type strings, this function
# tells you whether the port is an input, an output, or an inout.
#
#####
35 sub Get_Direction_From_Avalon_Type
{
    my ($crazy_type) = (@_);
    die "badly-formed port type specifier: $crazy_type" unless $crazy_type =~
        /(master|internal|external)_(input|output|inout)?(shared)?(.*)/;
40   return $2;
}

#####
45 # PTF_Eval_Expr
#
# If you have an assignment-value from a PTF-file,
# this subroutine tries to evaluate it as a numerical expression.
# This is handy for converting hex-values to "real numbers,"
50 # for example--or even allowing users to put honest-to-Pete
# arithmetic expressions in their PTF-files:
#
#   IRQ_Number = "36 + 0xC";      # Valid when evaluated.
#
55 # Sometimes, numeric fields can have special marker-values,
# like "N/A" or "peripheral-controlled". We give the caller option
# of specifying a list of such "special" values, which we return
# unmolested.
#
60 # The user may also optionally provide a description, which is
# handy for error-reporting.
#
#####
```

```

sub PTF_Eval_Expr
{
    my ($value, $description, @special_values) = (@_);

5    foreach $special (@special_values)
        { return $value if $value eq $special; }

    $description = "<unknown>" if $description eq "";

10    my $result = eval ($value);

    my $msg =<<EOM;
        Error: Could not evaluate the following expression:
            $value

15        This nasty little expression was found in the
            PTF data under this path:

            $description

20        When I tried to evaluate said nasty little expression,
            I got this here error:

            $@

25    EOM
        die $msg if $@;
        return $result;
    }

30    #####
    # PTF_Get_Boolean_Data_By_Path
    #
    # Fishes value out of PTF-file, then validates to make
    # sure it's boolean.
    #
35    #####
    sub PTF_Get_Boolean_Data_By_Path
    {
        my ($ptfRef, $path, $default) = (@_);
        my $data = get_data_by_path ($ptfRef, $path);

40        return &Vpp_Validate_Boolean ($data, $path, $default);
    }

45    #####
    # PTF_Get_And_Evaluate_Data_By_Path
    #
    # Gets indicated value out of PTF, then
    # tries to evaluate it using PTF_Eval_Expr, above.
    #
50    #####
    sub PTF_Get_And_Evaluate_Data_By_Path
    {
        my ($ptfRef, $path, @special_values) = (@_);
        my $data = &get_data_by_path (@_);

55        &PTF_Eval_Expr ($data, $path, @special_values);
    }

60    #####
    # PTF_Get_Required_Data_By_Path
    #
    # Just like ptf_parse's "get_data_by_path," except that

```

05220105-051201
TOP SECRET

```

# we print a nasty error message and die if the data
# we requested isn't there.
#
#####
5 sub PTF_Get_Required_Data_By_Path
{
    my ($ptfRef, $path, $error_message) = (@_);

    my $result = get_data_by_path($ptfRef, $path);

10    if ($result eq "")
    {
        my $ptfName = &get_name ($ptfRef) . " " . &get_data($ptfRef);
        my $msg =<<EOM;
15    Error: $error_message
        Required assignment:
        '$path'
        was not found in PTF section:
        '$ptfName'

20    EOM
        die $msg;
    }
    return $result;
25 }

#####
# PTF_Get_Required_Child_By_Path
#
# Just like ptf_parse's "get_child_by_path," except that
# we print a nasty error message and die if the child
# we requested isn't there.
#
#####
35 sub PTF_Get_Required_Child_By_Path
{
    my ($ptfRef, $path, $error_message) = (@_);

    my $result = get_child_by_path($ptfRef, $path);

40    if ($result eq "")
    {
        my $ptfName = &get_name ($ptfRef) . " " . &get_data($ptfRef);
        my $msg =<<EOM;
45    Error: $error_message
        Required PTF section:
        '$path'
        was not found in PTF section/file:
        '$ptfName'

50    EOM
        ribbit $msg;
    }
    return $result;
55 }

#####
# PTF_Build_Hash_From_Section
#
# Suppose you want to read -all- the data in some particular
# section of a PTF, and build a corresponding hash of values.
# That's exactly what this function does. It builds the hash,
# then returns a reference to it.
60

```

```

#
# This function is intended to gather data from "simple" sections
# (ones that don't contain other sections). If it encounters
# a child section, it displays a warning, and does the best it can
5 # with the result hash (the body of any child-section is ignored).
#
# If the section you asked for doesn't exist, this function
# dies with an error -- unless you specifically ask it not to.
#
10 # The "path" argument is optional--if you omit it, then
# we just convert "$ptfRef" into a hash.
#
#####
sub PTF_Build_Hash_From_Section
15 {
    my ($ptfRef, $path, $strict) = (@_);
    $strict = 1 if $strict eq "";

    my $db_Section = $ptfRef;
    if ($path ne "") {
20         if ($strict) {
            $db_Section = &PTF_Get_Required_Child_By_Path ($ptfRef, $path);
        } else {
            $db_Section = &get_child_by_path ($ptfRef, $path);
25         }
    }

    my %result_hash;
    my $num_children = &get_child_count ($db_Section);
    for (my $child_index = 0; $child_index < $num_children; $child_index++)
30 {
        my $db_Child = &get_child ($db_Section, $child_index);

        goldfish ("
35             PTF_Build_Hash_From_Section: encountered suspicious
            sub-section in $path.
            ") if &get_child_count ($db_Child) != 0;
        $result_hash{&get_name($db_Child)} = &get_data ($db_Child);
    }
40
    return \%result_hash;
}

#####
45 # PTF_New_Required_Ptf_From_File
#
# Just like ptf_parse's "new_ptf_from_file," except that
# we print a nasty error message and die if the file
# can't be opened.
50 #
#####
sub PTF_New_Required_Ptf_From_File
{
    my ($filename, $error_message) = (@_);
55
    my $result = new_ptf_from_file ($filename);

    if ($result eq "")
    {
60         my $msg =<<EOM;
        Error: $error_message
            Required PTF file:
            '$filename'
    }
}

```

09000106 "061201

could not be opened.

EOM

```
5      die $msg;
      return $result;
}

#####
10 # PTF_Check_Bool
#
# So you have a value in a hash. You got this value out of a PTF
# file, and you think it's a string representing a boolean
# setting (e.g. a string that says "TRUE" or "FALSE." Instead,
15 # you'd rather have a testable boolean value (or an error, if
# the string is something weird).
#
# That's what this function does.
#
20 #####
sub PTF_Check_Bool
{
    my ($hash_ref, $var_name, $default_value) = (@_);

25     $$hash_ref{$var_name} = &Vpp_Validate_Boolean ($$hash_ref{$var_name},
                                                    $var_name,
                                                    $default_value);
}

#####
30 # PTF_Allow
#
# So you have a value in a hash. You got this value out of a PTF
# file, and you think it's a string which can have one of only a few
35 # restricted values. Wouldn't it be nice to check?
#
# That's what this function does.
#
# If you pass-in a null value (i.e. the setting doesn't exist in the
40 # PTF-file) then this function will return without an error. Checking
# to be sure a setting has a legal value is different than checking
# to be sure a setting is, in fact, set.
#
#####
45 sub PTF_Allow
{
    my ($hash_ref, $var_name, @allowed_values) = (@_);

    my $value = $$hash_ref{$var_name};
    return if $value eq "";

    foreach $allowed (@allowed_values)
    { return if $value eq $allowed; }

50     die "Illegal value for $var_name setting: $value";
}

#####
60 # PTF_Eval
#
# So you have a value in a hash. You got this value out of a PTF
# file, and you think it's a string representing a numerical
# setting (e.g. a string that says "3" or "0x1ffff" Instead,
```

```

# you'd rather have the raw number (or an error, if
# the string is something weird.
#
# That's what this function does.
5  #
#####
sub PTF_Eval
{
10     my ($hash_ref, $var_name, @special_values) = (@_);
        $$hash_ref{$var_name} = &PTF_Eval_Expr ($$hash_ref{$var_name},
                                                $var_name,
                                                @special_values);
}
15 #####
# PTF_Require
#
# Given a hash which was extracted from a PTF-section, validates
20 # that the indicated member is present (non-NULL).  If not, it
# prints an error message.
#
#####
sub PTF_Require
25 {
    my ($hashref, $assignment_name) = (@_);
    return if $$hashref{$assignment_name} ne "";

    die "Required assignment ($assignment_name) not found in PTF.";
30 }
#####
# PTF_Default
#
# Given a hash which was extracted from a PTF-section, validates
35 # that the indicated member is present (non-NULL).  If it is NULL,
# then we give it a default value and print a polite warning.
#
#####
40 sub PTF_Default
{
    my ($hashref, $assignment_name, $default_value) = (@_);
    return if $$hashref{$assignment_name} ne "";

45     $$hashref{$assignment_name} = $default_value;

    print STDERR
        "Warning: PTF assignment ($assignment_name) missing.
        set to default value ($default_value)\n";
50 }
#####
# Add_Module_Ports_To_PTF
#
55 # Suppose you've just generated a module using Vpp.  Further suppose
# that this module has its very-own MODULE section in a PTF-file
# someplace.  Obviously, you want the PORT_WIRING section in said
# PTF-file section to agree with the actual Verilog ports on the
60 # module.
#
# Why, this very function right here takes your module (by name) and
# uses its' pre-declared "&List_Ports_For"-ports to create a correct

```



```

# PORT_WIRING section in the PTF section.
#
# You have to pass-in a reference to the PTF-section you want
# modified, as well as the name of the module whose ports you want
5 # to describe.
#
#####
sub Add_Module_Ports_To_PTF
{
10     my ($db_Module, $module_name) = (@_);

    my @port_list = &Get_Port_List ($module_name);

    # Error-checking. How polite.
15     scalar (@port_list) != 0 or die
        "Add_Module_Ports_To_PTF:
        No port list found for module named $module_name";

    &delete_child ($db_Module, "PORT_WIRING"); # Out with the old.
20
    foreach $port (@port_list) {
        foreach $attrib (&Get_Port_Attribute_List ($module_name, $port)) {

            next if $attrib eq "name"; # Already got the name, thanks.
25             &add_child_data ($db_Module,
                "PORT_WIRING/PORT $port/$attrib",
                &Get_Port_Attribute ($module_name, $port, $attrib)
            );
        }
30     } # END: foreach $port...
}

#####
# Add_Synthesis_Files_To_PTF
#
# Suppose you have a MODULE section in a PTF file, and you want
# to fill-in the HDL_INFO section with a list of HDL (probably Verilog) files.
#
# This is your function, pal. You just pass a reference to the PTF
40 # MODULE section and a list of filenames. We do the rest (which,
# truth to tell, isn't much).
#
#####
sub Add_Synthesis_Files_To_PTF
45 {
    my ($db_Module, @hdl_files) = (@_);

    &Add_HDL_INFO_Files_To_PTF ($db_Module,
50         "Synthesis_HDL_Files",
        @hdl_files);
}

#####
# Add_HDL_INFO_Files_To_PTF
#
# Suppose you have a MODULE section in a PTF file, and you want
# to fill-in the "HDL_INFO/<XXX>" assignment with a list of
# files.
#
60 # This is your function, pal. You just pass a reference to the PTF
# MODULE section and a list of filenames. We do the rest (which,
# truth to tell, isn't much).
#

```

FOOTNOTES

```
#####
sub Add_HDL_Info_Files_To_PTF
{
```

```
    my ($db_Module, $assignment_string, @file_list) = (@_);
```

```
    my $file_list_string = join (" ", @file_list);
    &add_child_data ($db_Module,
        "HDL_INFO/$assignment_string",
        $file_list_string
    );
```

```
}
```

```
#####
# Fill_In_Sections_And_Save_PTF_File
```

```
#
# Suppose you've just generated a module using Vpp. It's not
# a super-complicated module--it's just a simple module
# contained on one (1) simple verilog file with a simple set of
# ports.
```

```
#
# Further suppose you have a PTF-file with a MODULE section corresponding
# to the module you just now generated.
```

```
#
# You're responsible for filling-in these sections:
```

```
#     PORT_WIRING
#     HDL_INFO
```

```
#
# So that they accurately describe the module you just built.
# This function does exactly that, given nothing more than
# a reference to the PTF-file and another reference to the $args
# taken by your modlue (the very-same "$arg" hashref returned by
# &Process_Wizard_Script_Arguments).
```

```
#
# To do all this work for you, we make some assumptions:
```

```
#     1) As part of your Vpp-generation, you did a "&List_Ports_For"
#         describing your module's I/O pins.
```

```
#     2) Your module is named as-described in $$arg{name}.
```

```
#     3) It is defined in a VERILOG file of the same name (ending in
#         ".v", of course, and said file is in the $QUARTUS_PROJECT_DIR.
```

```
#     4) This one file is used for both simulation -and- synthesis.
```

```
#     5) There aren't any other HDL files, or any other trickiness.
```

```
#
# If so, this function does the generic "fill-in-the-blanks" work
# in the PTF-file. If your module needs something more complicated
# than this, then you can call some of the (rather obvious) sub-functions
# yourself with different arguments.
```

```
#
#####
sub Fill_In_Sections_And_Save_PTF_File
```

```
{
    my ($db_PTF_File, $arg, @HDL_file_list) = (@_);
```

```
    &Progress ("Updating PTF section for module: $$arg{name}.")
    if ($$arg{verbose});
```

```
    # if the user didn't supply a list of files, then we just use
```

```

# the obvious top-level module name for the entity under construction:
@HDL_file_list = ("$$arg(system_directory)/$$arg(name).v")
if (scalar (@HDL_file_list) == 0);

```

```

5 my $db_Module = &PTF_Get_Required_Child_By_Path
    ($db_PTF_File,
    "SYSTEM/MODULE $$arg(name)",
    "I could have sworn I saw $$arg(name) in
    here!");

```

```

10 &Add_Module_Ports_To_PTF ($db_Module, $$arg(name));
    &Add_Synthesis_Files_To_PTF ($db_Module, @HDL_file_list);
    &write_ptf_file ($db_PTF_File) or die
        "Couldn't write PTF File when generating $$arg(name)";
15 }

```

```

#####
# Is_Absolute_Path
#

```

```

20 # Return 1 (true) if it is, and 0 (false) if it ain't.
#
#####

```

```

sub Is_Absolute_Path

```

```

25 {
    my ($filename) = (@_);
    $filename =~ s|\\|\/|sg;
    $filename =~ s|^\\s+|sg;
    $filename =~ s|\\s+$|sg;

```

```

30 return 1 if $filename =~ /^\\/; # Starts with a slash.
    return 1 if $filename =~ /^[^\/]+\:/; # Some sort of drive-specifier.

```

```

    return 0;

```

```

35 }

```

```

#####
# Get_Base_Fname
#

```

```

40 # Given a filename which might contain a path, strip-off
    # the directory-part, leaving only the "base" filename part.
#
#####

```

```

sub Get_Base_Fname

```

```

45 {
    my ($filename) = (@_);

    $filename =~ /\.?*([^\\/]+)$/;
    my $base = $1;
    return $base;
50 }

```

```

1; # Modules must say "1"--mustn't they?

```

```

55

```

T02T90" 90T08860

mk_ram.pl

```

#####
# mk_ram.pl
#
#
# This Perl-script is the "Generator_Program"
# for the SOPC-Builder component class "altera_avalon_ram".
#
10 # This generator program is very similar to the one you'll find
# in the "altera_avalon_uart" library directory. If you are interested
# in reading an overlong comment which describes the operation
# of, and philosophy behind, the standard set of generator programs,
# I strongly suggest that you review the uart's generator program.
15 # It is in a file named "mk_uart.pl"
#
use wiz_convert;
use wiz_utils;

20 #####
# Mk_RAM
#
# Builds a Nios on-chip RAM from a list of named arguments.
#
25 $Mk_RAM_Doc=<<<END_OF_DOCUMENTATION_STRING ;
# LONG NAME      SHORT NAME    DEFAULT      DESCRIPTION
# -----
# Contents      --none--      blank        *(blank|germs|user_file)* init?
# Writeable     writeable     1            *boolean*   RAM or ROM?
30 # Initfile     file          --none--     Either .srec or .mif from user.
END_OF_DOCUMENTATION_STRING
#
#
#####
35 sub Mk_RAM
{
    my ($arg, $user_defined, $db_Module, $db_PTF_File) =
        &Process_Wizard_Script_Arguments ($Mk_RAM_Doc, @_);

40    &Progress ("Generating logic for: $$arg{name}.");

    my $SBI = &PTF_Build_Hash_From_Section ($db_Module, "SYSTEM_BUILDER_INFO");
    &PTF_Eval ($SBI, "Address_Width");
    &PTF_Eval ($SBI, "Data_Width");
45    &PTF_Eval ($SBI, "Address_Span");

    # We have to poke-around in the SYSTEM's WSA section to find
    # out whether we should create a simulation model or not:
    #
50    my $Sys_WSA = &PTF_Build_Hash_From_Section
        ($db_PTF_File, "SYSTEM/WIZARD_SCRIPT_ARGUMENTS");

    #####
55    # A bit of validation to make sure the number of address bits
    # is sensible for the given memory size:
    #
    my $byte_size = $$SBI{Address_Span};
    my $max_byte_size = 2**$$SBI{Address_Width} * ($$SBI{Data_Width} / 8);

60    $byte_size <= $max_byte_size or die "
        Address_Width ($$SBI{Address_Width}) too small for memory
        $$arg{name} with $byte_size bytes\n";

```



```

$convert_cmd .= " $contents_file_name";
$convert_cmd .= " $mif_name";
$convert_cmd .= " --width=$lane_width";
$convert_cmd .= " --lanes=$num_lanes";

```

```

5   my $error_code = &System_Win98_Safe ($convert_cmd);
    if ($error_code != 0) {
        print STDERR "      Error converting initialization file for
10  $$arg{name}\n";
        print STDERR "      $contents_file_name\n";

        if ($$arg{Contents} =~ /germs/i) {
            die " (Perhaps the Nios SDK is not installed?)\n";
        }
15     else {
        print STDERR "      To build a 'placeholder' memory with no init
file,\n";
        print STDERR "      specify 'Blank' in the On-Chip Memory
MegaWizard.\n";
20     die;
    }
}

# The end result of all that fooling-around is either:
25 #   * A single mif-file named "$mif_name."
#   * Multiple mif-files, one-per-lane.
#
# Either way, we pedantically make a "list" out of it/them.
#
30 my @miffile_list = ();

if (!$$arg{Writeable}) {
    push (@miffile_list, $mif_name);
} else {
35     my $base_name = $mif_name;
    $base_name =~ s/\.mif$/i;
    for (my $i = 0; $i < $num_lanes; $i++) {
        push (@miffile_list, "$base_name\_lane_$i.mif");
    }
40 }

#####
# Convert all that crapola to .dat-files, if simulation is
# required. Now you see why, above, we made a list of all MIF-files,
45 # in the ROM case, when there was only one file.
#
# Everything else in here handles our contents-file by its full
# or relative path (whatever $arg{system_directory} is). But
# here, we do something special to "hide away" the .dat-files
50 # in their own secret directory.
#
my %datfile_hash; # Save dat-file names in here, indexed by mif-file.
if ($$Sys_WSA{do_build_sim}) {
    &Progress ("Converting simulation file(s) for $$arg{name}")
55     if $$arg{verbose};

    foreach $mif_file (@miffile_list) {
        my $dat_name = $mif_file;
        $dat_name =~ s/.*?([^\s\/]+)$/1/; # Strip-off leading path
60         $dat_name =~ s/\.mif/\.dat/i;    # Change extension

        &Convert_Mif_To_Dat ($mif_file, "$$arg{system_sim_dir}/$dat_name");
        $datfile_hash{$mif_file} = $dat_name;
    }
}

```

```

    }
}

my $stop_file = "$$arg(system_directory)/$$arg(name).v";
my @synth_file_list = ($stop_file);

#####
# Now run vpp.
#
# notice that we build two entirely-different things,
# depending on whether this is a RAM or a ROM.
#
# That is a true fact. You might have done it differently.
# but -you- were comfortably asleep in your nice, warm bed
# while -I- was doing it.
if ($$arg(Writable)) {
    # Run Vpp several times to make the byte-lane modules.
    # Regrettably, there must be multiple separate black-boxes
    # because the synthesis tool doesn't handle parameterization
    # correctly and, obviously, each lane has a different init-file
    # parameter. And since there is one black-box per lane, there
    # needs to be one -file- per lane, because Quartus resolves
    # (fills-in) black-boxes by -filename.-
    #
    for (my $i = 0; $i < scalar (@miff_file_list); $i++) {
        my $mif_file = $miff_file_list[$i];
        my $lane_module_name = "$$arg(name)_lane_$i";
        my $lane_file_name = "$$arg(system_directory)/$lane_module_name.v";
        push (@synth_file_list, $lane_file_name);

        &Vpp ("-Q",
            "-O", "$lane_file_name",
            "ONCHIP_RAM_LANE_MODULE_NAME = $lane_module_name",
            "ONCHIP_RAM_WIDTH = $$SBI{Data_Width}",
            "ONCHIP_RAM_DEPTH = $depth",
            "ONCHIP_RAM_MIF_FILE = $mif_file",
            "ONCHIP_RAM_DAT_FILE = $datfile_hash($mif_file)",
            "-H", "$$arg(sopc_directory)/bin/vpp/generator_functions.vpp",
            "$$arg(class_directory)/onchip_ram_byte_lane.vpp",
        );
    }

    # Making a RAM requires both byte-lanes and a box to put them
    # in. Run Vpp again to build the top-level module.
    &Vpp ("-Q",
        "-O", "$stop_file",
        "ONCHIP_RAM_MODULE_NAME = $$arg(name)",
        "ONCHIP_RAM_WIDTH = $$SBI{Data_Width}",
        "ONCHIP_RAM_DEPTH = $depth",
        "-H", "$$arg(sopc_directory)/bin/vpp/generator_functions.vpp",
        "$$arg(class_directory)/onchip_ram.vpp",
    );
} else {
    # You can make a ROM with only one module:
    my $dat_filename = $datfile_hash($miff_file_list[0]);
    &Vpp ("-Q",
        "-O", "$stop_file",
        "ONCHIP_ROM_MODULE_NAME = $$arg(name)",
        "ONCHIP_ROM_DEPTH = $depth",
        "ONCHIP_ROM_WIDTH = $$SBI{Data_Width}",
        "ONCHIP_ROM_MIF_FILE_NAME = $mif_name",
        "ONCHIP_ROM_DAT_FILE_NAME = $dat_filename",
    );
}

```

09830106 "061201

```

"-H", "$arg{sopc_directory}/bin/vpp/generator_functions.vpp",
"$arg{class_directory}/onchip_rom.vpp",
);

```

```

5      # We need to list our MIF-files, so their names can be "crushed" if
      #   required for MaxPlus+II(two).
      #
10     &Add_HDL_INFO_Files_To_PTF
        ($db_Module, "MIF_Files", @miffile_list);

        &Fill_In_Sections_And_Save_PTF_File ($db_PTF_File, $arg, @synth_file_list);
    }

15     #####
    # &Generate_Blank_Mif_File
    #
    # This function, um, generates a blank...mif file.
    #
20     # You pass-in the $arg-hash (ref) from mk_RAM, so we have all the
    # information we might ever need.  To generate a blank mif-file,
    # you don't need that much info.
    #
    #####
25     sub Generate_Blank_Mif_File
    {
        my ($arg, $SBI, $depth) = (@_);

        my $mif_header =<<EOM;
30     /* This source file generated by mk_ram.pl (&Generate_Blank_Mif_File)*/
        /* This file is used to initialize the memory-type module named:      */
        /*      $$arg{name}                                                    */
        WIDTH=$$SBI(Data_Width);
        DEPTH=$depth;
35     ADDRESS_RADIX=UNS;
        DATA_RADIX=HEX;
        CONTENT BEGIN

        EOM

40     my $blank_mif_name = "$$arg{system_directory}/$$arg{name}_blank.mif";

        open (MIFOUT, "> $blank_mif_name") or die
            "Couldn't open $blank_mif_name: $!";

45     print MIFOUT $mif_header;
        for (my $i=0; $i < $depth; $i++) {
            print MIFOUT "$i : 0;\n";
        }

50     print MIFOUT "END;\n";
        close (MIFOUT);

        return $blank_mif_name;
55 }

    #####
    # &Check_For_Other_Germs_Monitors
    #
60     # Riffle through the PTF-file and see if there are any other
    # GERMS-monitor-bearing memories.  If there are, emit a warning.
    # There must be only one.
    #

```



```
#####  
# Execution begins here  
#####  
5 &Mk_RAM (@ARGV);
```

09880106 "061201

mk_pio.pl

#####

mk_pio.pl

This Perl-script is the "Generator_Program"
for the SOPC-Builder component class "altera_avalon_pio".
#

This generator program is very similar to the one you'll find
in the "altera_avalon_uart" library directory. If you are interested
in reading an overlong comment which describes the operation
of, and philosophy behind, the standard set of generator programs,
I strongly suggest that you review the uart's generator program.
It is in a file named "mk_uart.pl"

use wiz_utils;

#####

Mk_PIO

Builds a Nios PIO peripheral from a list of named arguments.

\$Mk_PIO_Doc=<<END_OF_DOCUMENTATION_STRING ;

LONG NAME SHORT NAME DEFAULT DESCRIPTION

has_tri tri NO *boolean* Tristatable pins?
has_out out YES *boolean* Dedicated output pins?
has_in in YES *boolean* Dedicated output pins?
edge_type edge NONE *(NONE|RISING|FALLING|ANY)*
irq_type irq NONE *(NONE|LEVEL|EDGE)* Type of IRQ?

END_OF_DOCUMENTATION_STRING

#

#

#####

sub Mk_PIO

{

my (\$arg, \$user_defined, \$db_Module, \$db_PTF_File) =
&Process_Wizard_Script_Arguments (\$Mk_PIO_Doc, @_);

my \$SBI = &PTF_Build_Hash_From_Section (\$db_Module, "SYSTEM_BUILDER_INFO");

&Vpp ("-Q",
"-O", "\$arg(system_directory)/\$arg(name).v",
"PIO_MODULE_NAME = \$arg(name)",
"PIO_MODULE_NAME = \$arg(name)",
"PIO_TRISTATE_PINS = \$arg(has_tri)",
"PIO_OUTPUT_PINS = \$arg(has_out)",
"PIO_INPUT_PINS = \$arg(has_in)",
"PIO_EDGE_CAPTURE = \$arg(edge_type)",
"PIO_INTERRUPT = \$arg(irq_type)",
"PIO_BITS = \$\$SBI(Data_Width)",
"-H",
"\$arg(sopc_directory)/bin/vpp/generator_functions.vpp",
"\$arg(class_directory)/pio_core.vpp",
);

&Fill_In_Sections_And_Save_PTF_File (\$db_PTF_File, \$arg);

}

#####

Execution begins here

#####

&Mk_PIO (@ARGV);

THE UNIVERSITY OF CHICAGO

mk_spi.pl

#####

mk_spi.pl

This Perl-script is the "business end" of the
Nios SPI Wizard. The Wizard itself is a GUI-layer
which quizzes the user and passes his(her) choices
along to this very script.

The kind of spi core we build depends on the
parameters we get. The parameters are "named arguments,"
Named arguments are one long comma-delimited string,
a list of 'normal' command-line arguments, or any combination
of both (we just smash all the command-line arguments together
into one long string anyhow).

The comma-delimited elements have the form:
<arg_name> = <value>.

For a list of all the argument-names and their allowed values,
see the table below.

use wiz_utils;

mk_spi

Builds a Nios spi interface from a list of named arguments.

\$mk_spi_Doc=<<END_OF_DOCUMENTATION_STRING ;

LONG NAME SHORT NAME DEFAULT DESCRIPTION

#	-----	-----	-----	-----
	databits	--none--	8	*(\d+)* number of data bits.
	clockdiv	--none--	--none--	*(\d+)* SCLK-gen divisor.
	numslaves	--none--	1	*(\d+)* Number of slave devices.
	ismaster	--none--	1	*(0 1)* If 1, master else slave.
	clockpolarity	--none--	0	*(0 1)* If 0, clock idles low.
	clockphase	--none--	0	*(0 1)* (see note, below).
	lsbfirst	--none--	0	*(0 1)* 0 --> data is MSB first.
	targetclock	--none--	1	User's target SCLK frequency (Hz).
	targetssdelay	--none--	1	User's target delay after SS_n (s).
	extradelays	--none--	0	*(0 1)* Boolean - if 0, no extra delay.

END_OF_DOCUMENTATION_STRING

The "clockphase" argument:
If 0, data is sampled on transition-to-idle of SCLK.

The "delayafterss" argument:
Delay after SS_n before data transmission starts, in SCLK half-cycles.

Note: delayafterss is a historical relic. In these modern times,
spi_core.vpp

calculates its own delayafterss value, using the system clock rate and the
user's targetssdelay value.

#####

sub mk_spi

{
my (\$arg, \$user_defined, \$db_Module, \$db_PTF_File) =
&Process_Wizard_Script_Arguments (\$mk_spi_Doc, @_);

&Progress ("Generating logic for: \$\$arg(name).");

The system clock rate is part of the equations that generate
actual SCLK and delay-after-ss values from the user's target
values. It's easy to read the ptf file here, so grab the clock
rate and pass it along.

my \$clock_freq = &PTF_Get_Required_Data_By_Path (\$db_PTF_File,
"SYSTEM/WIZARD_SCRIPT_ARGUMENTS/clock_freq",
"System clock frequency not specified");

&Vpp ("-Q",
"-O", "\$\$arg(system_directory)/\$\$arg(name).v",
"SPI_MODULE_NAME = \$\$arg(name)",
"DATABITS = \$\$arg(databits)",
"NUMSLAVES = \$\$arg(numslaves)",
"ISMASTER = \$\$arg(ismaster)",
"CPOL = \$\$arg(clockpolarity)",
"CPHA = \$\$arg(clockphase)",
"LSBFIRST = \$\$arg(lsbfirst)",
"INPUT_CLOCK = \$clock_freq",
"TARGETCLOCK = \$\$arg(targetclock)",
"TARGETSSDELAY = \$\$arg(targetssdelay)",
"EXTRADELAY = \$\$arg(extradelay)",
"-H", "\$\$arg(sopc_directory)/bin/vpp/generator_functions.vpp",
"\$\$arg(class_directory)/spi_core.vpp",
);

&Fill_In_Sections_And_Save_PTF_File (\$db_PTF_File, \$arg);

}

Execution begins here

&mk_spi (@ARGV);

mk_timer.pl

#####

mk_timer.pl

#

This Perl-script is the "Generator_Program"

for the SOPC-Builder component class "altera_avalon_pio".

#

This generator program is very similar to the one you'll find

in the "altera_avalon_uart" library directory. If you are interested

in reading an overlong comment which describes the operation

of, and philosophy behind, the standard set of generator programs,

I strongly suggest that you review the uart's generator program.

It is in a file named "mk_uart.pl"

#

use wiz_utils;

#####

Mk_Timer

#

Builds a Nios timer from a list of named arguments.

\$Mk_Timer_Doc=<<END_OF_DOCUMENTATION_STRING ;

LONG NAME SHORT NAME DEFAULT DESCRIPTION

-- At present, Timers are just timers. They don't really

depend on any timer-specific arguments. Howd'ya like that? --

END_OF_DOCUMENTATION_STRING

#

#

#####

sub Mk_Timer

{

my (\$arg, \$user_defined, \$db_Module, \$db_PTF_File) =

&Process_Wizard_Script_Arguments (\$Mk_Timer_Doc, @_);

&Progress ("Generating logic for: \$\$arg{name}.");

&Vpp ("-Q",

"-O", "\$\$arg{system_directory}/\$\$arg{name}.v",

"TIMER_MODULE_NAME = \$\$arg{name}",

"-H", "\$\$arg{sopc_directory}/bin/vpp/generator_functions.vpp",

"\$\$arg{class_directory}/timer.vpp",

);

&Fill_In_Sections_And_Save_PTF_File (\$db_PTF_File, \$arg);

}

#####

Execution begins here

#####

&Mk_Timer (@ARGV);

```

mk_uart.pl

#####
# mk_uart.pl
#
# This Perl-script is the "Generator_Program"
# for the SOPC-Builder component class "altera_avalon_uart".
#
# This program (Perl script) is launched by the
# SOPC-builder wizard as part of the "generation" phase.
# "Generation" takes place after the user presses the wizards'
# "Finish" button, but before synthesis takes place. During this
# time, HDL-files for all components in the system are created.
# Each modules' HDL-files are (one presumes)
# generated by its "Generator_Program." This here is the
# generator-program what generates UARTS.
#
# You may sensibly ask: How do we know what kind of UART to
# generate, given that there are oh-so-many options and parameters?
# Easy. The specification for the UART we are to build is
# contained in the system's PTF-file.
#
# We can find the PTF-file by using our command-line arguments.
# All generator-programs run by the SOPC-builder get the same
# set of command-line arguments, which are:
#
# --system_name=<sys_name>
#
# Gives the name of the system module which will contain the
# module we're supposed to generate. We need the name of the system
# (and the directory, above) so that we can find the system's
# PTF-file. We can't do Jack without the PTF file.
#
# -system_directory=<system-directory-path>
#
# Gives the name of the directory where we are to generate
# our Verilog files, and in which the system PTF-file can be found.
# This is typically the users' quartus project directory.
#
# -target_module_name=<mod-name>
#
# Gives the name of the module in the system PTF file which
# we are supposed to generate. This lets us extract the appropriate
# options, parameters, and whatnot from the PTF-file so that
# we generate the right thing, and give it all the right name
# when we're done.
#
# --sopc_directory=<lib_dir>
#
# Gives the full path name for the SOPC-Builder installation
# directory. From here, we can find all the relevant SOPC
# Perl-scripts and utility programs (in the ...\bin subdirectory).
# We can also find any and all installed library components
# (by-name in the ...\components subdirectory) including,
# significantly, the library for the very component we're trying to
# generate. We must know where this library component
# and all its support files reside (including, for example, its
# "class.ptf" file). This makes it easy to find all the
# "raw materials" we need to create the module that this
# "generator_program" is supposed to generate.
#
# You may notice that most of the work performed by this

```



```

# generator program is done by handily-dandily library functions.
# Indeed, the "wiz_utils" Perl library (.pm module) contains functions
# which are a boon to the generator_program author. Here is a
# brief description of the library functions used by this
5 # generator program. I list them here in the fond hope that they
# might be useful to someone else trying to create their own
# generator program.
#
# This generator program uses the utility functions:
10 #
# &Process_Wizard_Script_Arguments
#
# (from the library "wiz_utils.pm") This function takes our
# input arguments (listed above) and uses them to:
15 #
# * Find and open the system PTF file.
#
# * Extracts this modules' WIZARD_SCRIPT_ARGUMENTS section.
#
# * Reads all the arguments and verifies them against a document
20 # which describes what we expect
# (see DOCUMENTATION_STRING, below).
#
# * Passes-back all the arguments in a hashref we lovingly
# refer to as $arg.
#
# * Hands us back a reference to the freshly-opened PTF-file
#
# * Sets a lot of handy global variables, like $QUARTUS_PROJECT_DIR.
#
# In short, this function does just about everything we need to do
# to find and read this modules' parameters out of the system PTF file.
#
# &Fill_In_Sections_And_Save_PTF_File
35 #
# (from the library "wiz_utils.pm") You run this program after
# you have generated all the logic (HDL) for your module. This
# function fills-in your modules' PTF-section with lots of useful
# information. In particular, it adds data to the PORT_WIRING
40 # section which agrees with the actual ports present on your
# module. This only works, of course, if your module logic was
# generated by VPP.
#
# And I hear you asking: "What, exactly, is Vpp?"
45 #
# Ah, yes. Vpp. Vpp is a Verilog/Perl hybrid, the purpose of which
# is to generate plain-vanilla Verilog output. Vpp is a very powerful
# way to generate lots of parameterized, conditionalized, itemized,
# and sanitized logic from good-old Perl, the language we all love so
50 # much. If you want to learn all about Vpp, there is an admirably-long
# comment atop the library file "vpp.pm" With that, I leave you
# to your own spunk and initiative to learn about Vpp.
#
#
55 use wiz_utils;

#####
# Mk_Uart
#
60 # Builds a Nios uart from a list of named arguments.
# These arguments are actually fished-out from the
# WIZARD_SCRIPT_ARGUMENTS section of the system PTF file.
# We can find the system PTF file by following the trail

```

```

# of breadcrumbs provided in our actual, official command-line
# arguments ("-directory" and such) described above.
#
5 $Mk_Uart_Doc=<<END_OF_DOCUMENTATION_STRING ;
# LONG NAME      SHORT NAME      DEFAULT      DESCRIPTION
# -----
    baud          baud          115200       Uart baud rate.
    fixed_baud     fix_b         1            *boolean* Delete baud register?
10    parity        par          N            *(N|E|O|S1|S0)* Parity type.
    data_bits     bits          8            *(8|7)* Number of data bits
    stop_bits     stop          2            *(1|2)* Number of stop bits.

END_OF_DOCUMENTATION_STRING
#
15 #
#####
sub Mk_Uart
(
20     my ($arg, $user_defined, $db_Module, $db_PTF_File) =
        &Process_Wizard_Script_Arguments ($Mk_Uart_Doc, @_);

        &Progress ("Generating logic for: $$arg{name}.");

25     # The only required information which isn't supplied to the
    # uart directly in its own PTF-section is the system clock
    # frequency. We just have to rummage around in the system's
    # top-level PTF data to get that:

    my $clock_freq = &PTF_Get_Required_Data_By_Path ($db_PTF_File,
        "SYSTEM/WIZARD_SCRIPT_ARGUMENTS/clock_freq",
        "System clock frequency not specified");

30     &Vpp ("-Q",
        "-O", "$$arg{system_directory}/$$arg{name}.v",
        "UART_CORE_MODULE_NAME      = $$arg{name}",
        "UART_BAUD_RATE              = $$arg{baud}",
        "UART_FIXED_BAUD_RATE        = $$arg{fixed_baud}",
        "UART_PARITY                  = $$arg{parity}",
        "UART_DATA_BITS               = $$arg{data_bits}",
        "UART_STOP_BITS               = $$arg{stop_bits}",
        "UART_INPUT_CLOCK_FREQ       = $clock_freq",
        "-H", "$$arg{sopc_directory}/bin/vpp/generator_functions.vpp",
        "$$arg{class_directory}/uart_core.vpp",
        "$$arg{class_directory}/rx.vpp",
45     "$$arg{class_directory}/tx.vpp",
        );

        &Fill_In_Sections_And_Save_PTF_File ($db_PTF_File, $arg);

50     }

#####
# Execution begins here
55 #####
&Mk_Uart (@ARGV);

```

09300106051001
TOTAL

mk_usersocket.pl

#####

mk_usersocket.pl

This Perl-script is the "business end" of the
Nios User Socket Wizard. The Wizard itself is a GUI-layer
which quizzes the user and passes his(her) choices
along to this very script.

The kind of user socket we build depends on the
parameters we get. The parameters are "named arguments,"
Named arguments are one long comma-delimited string,
a list of 'normal' command-line arguments, or any combination
of both (we just smash all the command-line arguments together
into one long string anyhow).

The comma-delimited elements have the form:

<arg_name> = <value>.

For a list of all the argument-names and their allowed values,
see the table below.

use wiz_utils;

#####

Mk_UserSocket

Under the new "one-PTF" regime, user-socket wizards are a bit of an
odd duck. The "Generator Program" (the very script you're reading
now) doesn't actually -do- much, because there's no logic
(HDL-content) associated directly with the user-defined interface.
The interface is just a bunch of ports.

So, our responsibility, really, is to fill-in the relevant PORT_WIRING
section in the PTF-file.

Notice that, for user-sockets, the Java wizard is responsible for
all of the assignments in the SYSTEM_BUILDER_INFO section--as it
must be, or else the sopc-builder table wouldn't know what to do
with this thing (SBI must be complete when the sub-wizard exits).

Consequently, the user-socket wizard's method of telling us (the
generator program) what to do is through the SBI-settings
themselves.

If you choose to look at it this way, the entire -point- of the
user-socket java-wizard is to provide a GUI for all the SBI settings.

my \$pc = "peripheral_controlled";

Builds a Nios timer from a list of named arguments.

\$Mk_UserSocket_Doc=<<END_OF_DOCUMENTATION_STRING ;

LONG NAME SHORT NAME DEFAULT DESCRIPTION

keep_legacy_ports legacy 0 Don't generate new port list.

END_OF_DOCUMENTATION_STRING

#

#

#####

sub Mk_UserSocket

{

```
my ($arg, $user_defined, $db_Module, $db_PTF_File) =
    &Process Wizard Script Arguments ($Mk UserSocket_Doc, @_);
```

```
&Progress ("Generating port list for: $$arg{name}.");
```

```
my $SBI = &PTF_Build_Hash_From_Section ($db_Module, "SYSTEM_BUILDER_INFO");
&PTF_Eval ($SBI, "Address_Width");
&PTF_Eval ($SBI, "Data_Width");
```

```
if (($$arg{keep_legacy_ports})) {
    &List_Ports_For_User_Socket ($arg, $SBI);
    &Add_Module_Ports_To_PTF($db_Module, $$arg{name});
    &write_ptf_file ($db_PTF_File) or die
        "Couldn't write PTF File when generating $$arg{name}";
}
```

}

#####

```
# List_Ports_For_User_Socket
```

#

```
# Builds-up a string that describes the socket's ports, then
# calls "&List_Ports_For".
```

```
# Uses, mostly, information out of the SYSTEM_BUILDER_INFO section
# to figure-out what ports this thing has.
```

#

#####

```
sub List_Ports_For_User_Socket
```

{

```
my ($arg, $SBI) = (@_);
```

[illegible]

```
my $data is shared    = $$SBI{Uses_Tri_State_Data_Bus};
```

```
my $control_is_shared = ($data_is_shared) &&  
($$SBI{Setup_Time} == 0) &&  
($$SBI{Hold_Time} == 0) ;
```

```
my $irq_port    = "irq"    | 1 | 0 | avalon_role = irq";
my $wait_port   = "wait"   | 1 | 0 | avalon_role = waitrequest";
```

```
my @unshared_ports = (
    "chipselect" | 1 | I | avalon_role = chipselect");
```

```
my @shared_with_data_ports = (
    "byteenablen" | $be_width | I | avalon_role = byteenablen",
    "address" | $A_bits | I | avalon_role = address" );
```

```
my @sometimes_shared_control_ports = (
    "written" | 1 | I | avalon_role = written",
    "readn" | 1 | I | avalon_role = readn" );
```

```
if ($$SBI(Uses_Tri_State_Data_Bus)) {
    $data_is_shared = 1;
    push (@shared_with_data_ports,
        "data          | $D_bits          | inout | avalon_role = data");
} else {
    push (@unshared_ports,
        "readdata       | $D_bits          | 0      | avalon_role = readdata",
```

25
00000106-061201
30

```
        "writedata | $D_bits      | I      | avalon_role = writedata");
    }

5    push (@unshared_ports, $irq_port) if $$SBI{Has_IRQ};
    push (@unshared_ports, $wait_port) if $has_wait_pin;

    # Add "shared" attribute to all ports that might need it:
    #
10   for (my $i=0; $i < scalar(@shared_with_data_ports); $i++) {
        $shared_with_data_ports [$i] .= " | is_shared = $data_is_shared";
    }

    for (my $i=0; $i < scalar(@sometimes_shared_control_ports); $i++) {
15     $sometimes_shared_control_ports [$i] .= " | is_shared= $control_is_shared";
    }

    my @port_list = (@unshared_ports,
                     @shared_with_data_ports,
20     @sometimes_shared_control_ports,
                     );

    &List_Ports_For($$arg{name}, @port_list);
}

#####
# Execution begins here
#####
30 &Mk_UserSocket (@ARGV);
```

mk_nios.pl

#####

mk_nios.pl

#

This Perl-script is the "Generator_Program"
for the SOPC-Builder component class "altera_nios_cpu".

This generator program is very similar to the one you'll find
in the "altera_avalon_uart" library directory. If you are interested
in reading an overlong comment which describes the operation
of, and philosophy behind, the standard set of generator programs,
I strongly suggest that you review the uart's generator program.
It is in a file named "mk_uart.pl"

use wiz_convert;

use wiz_utils;

#####

#

Mk_Nios

#

Builds a Nios core from named arguments, including all
design and support files, plus synthesis script.

\$Mk_Nios_Doc=<<END_OF_DOCUMENTATION_STRING ;

#	LONG NAME	SHORT NAME	DEFAULT	DESCRIPTION
#	num_regs	regs	25	*(128 256 512)* Reg file size.
#	shift_size	shift	7	*(1 3 7 15 31)* Bits/clock shift speed
#	mstep	mstep	1	*boolean* Include MSTEP unit?
#	multiply	multiply	0	*boolean* Include multiply unit?
#	wvalid_wr	wr_wv	0	*boolean* Writeable WVALID reg?
#	rom_decoder	--none--	1	*boolean* Decoders in ROM?
#	*reset_module	--none--	--none--	CPU reset vector lives in here.
#	reset_offset	--none--	0	Reset vec offset from device base
#	*vecbase_module	--none--	--none--	CPU vector table lives in here.
#	vecbase_offset	--none--	0	Vecbase this far from device base.
#	mm_span	mm_span	--none--	main memory address span.
#	mm_base	mm_base	--none--	main memory base address.

END_OF_DOCUMENTATION_STRING

#

#

#####

sub Mk_Nios

{

my (\$arg, \$user_defined, \$db_Module, \$db_PTF_File) =
 &Process_Wizard_Script_Arguments (\$Mk_Nios_Doc, @_);

Allow expressions, hex-numbers, etc. for the address-offsets

#

&PTF_Eval (\$arg, "reset_offset");

&PTF_Eval (\$arg, "vecbase_offset");

&Progress ("Generating Nios CPU: \$\$arg{name}.");

#####

We'll be rummaging-around in the system PTF file quite a
lot, so a handle to the "SYSTEM"-section and some of its important
children would be useful.

my \$db_Sys = &PTF_Get_Required_Child_By_Path (\$db_PTF_File,
 "SYSTEM");

5
10
15
20
25
30
35
40
45
50
55
60

```
my $SBI      = &PTF_Build_Hash_From_Section ($db_Module,  
                                             "SYSTEM_BUILDER_INFO");  
my $Sys_WSA = &PTF_Build_Hash_From_Section ($db_Sys,  
                                             "WIZARD_SCRIPT_ARGUMENTS");  
5  &PTF_Check_Bool ($Sys_WSA, "do_build_sim", 0);  
    &PTF_Default ($Sys_WSA, "device_family", "APEX20K");  
  
#####  
10 # Get globals that will control synthesis and place and route  
    #  
    my @globals_2_control_synthesis_place_and_route =  
        &fillup_synthesis_place_and_route_globals ($db_Module);  
  
15 #####  
    # Start a 'rummagin'.  
    #  
    # Go through all the modules in the system and see if any have a  
    # "registered chip select." If (and only if) any module does, then  
20 # the CPU needs to be built with the "idle_cycle" option set to "YES"  
    #  
    # While we're looping through every module, now would also be a good  
    # time to pull-out the base-addresses of the modules designated  
    # as the reset-module and the vecbase-module, respectively.  
    #  
    my $reset_base_address      = 0;  
    my $vec_table_base_address = 0;  
    my $do_use_idle_cycle       = 0;  
  
30 my $num_children = &get_child_count($db_Sys);  
    for (my $child_index = 0; $child_index < $num_children; $child_index++) {  
  
        my $db_Module = &get_child ($db_Sys, $child_index);  
        next unless &get_name ($db_Module) eq "MODULE";  
        my $module_name = &get_data ($db_Module);  
  
        # If this is the designated reset- (or vecbase-) module,  
        # remember the base-address.  
        #  
40 if ($module_name eq $$arg{reset_module}) {  
            $reset_base_address = &PTF_Get_And_Evaluate_Data_By_Path  
                ($db_Module, "SYSTEM_BUILDER_INFO/Base_Address");  
        }  
        if ($module_name eq $$arg{vecbase_module}) {  
5  $vec_table_base_address = &PTF_Get_And_Evaluate_Data_By_Path  
            ($db_Module, "SYSTEM_BUILDER_INFO/Base_Address");  
        }  
  
        # If this module has any registered chip-selects, then there  
        # will be a special assignment to this effect in its SBI-section:  
        #  
50 $do_use_idle_cycle = 1  
            if &PTF_Get_Boolean_Data_By_Path ($db_Module,  
                "SYSTEM_BUILDER_INFO/Uses_Registered_Select_Signal",  
55 "FALSE");  
    }  
  
    my $vecbase_address = $vec_table_base_address + $$arg{vecbase_offset};  
    my $reset_address = $reset_base_address + $$arg{reset_offset};  
  
60 # For historical reasons, the Nios VPP-stuff wants the number of  
    # address bits converted to the "highest system address." Fine.  
    my $highest_address = (2**$SBI{Address_Width}) - 1;
```

```

#####
# User-instructions.
#
# Read and interpret any USER_INSTRUCTION sections
# of the CPU modules' PTF. Here's an example of a "USER_INSTRUCTION"
# section:
#
#       USER_INSTRUCTION    MAD_FMUL
#       {
#           format="RR";
#           opcode="011100";
#           uses_prefix="1";
#           has_sequential_logic="1";
#           top_module_name = "MAD_FMUL_Unit";
#           do_synthesize_top="1";
#
#       HDL_INFO
#       {
#           synthesis_files = "fmul_implementation.v";
#       }
#
# Information about the instructions we find is (evilly) passed
# to the Nios generator-programs through (gasp!) global variables
# They have scary-looking all-caps names that start with "NIOS",
# so it should be pretty hard to get into trouble.
#
# Also, now's the time when we start accumulating HDL-files
# for synthesis, if any.
#
# NOTE: This feature is in a sort of pre-release limbo. It's here,
# but we're not really telling anyone about it (if you're reading
# this comment, consider yourself a lucky winner). For this reason,
# error-checking is pretty minimal. It would be easy to make
# an invalid USER_INSTRUCTION section that would crash the
# Nios-generation step. Caveat usor.
#
# Note that this time we're looping-through the Nios's sub-sections,
# instead of the system's sub-sections:
#
$NIOS_HAS_USER_INSTRUCTIONS = 0;
$NIOS_USER_INSTRUCTION_SUPPORT_PREFIX = 0;
undef %NIOS_USER_INSTRUCTION_FORMAT;    # Null-out our globals
undef %NIOS_USER_INSTRUCTION_OPCODE;
undef %NIOS_USER_INSTRUCTION_GETS_PREFIX;
undef %NIOS_USER_INSTRUCTION_IS_SEQUENTIAL;
undef %NIOS_USER_INSTRUCTION_MODULE_NAME;
undef %NIOS_USER_INSTRUCTION_IS_BLACK_BOX;
@NIOS_USER_MNEMONIC_LIST = ();

my @Instruction_Unit_Synth_File_List = ();

my $num_children = &get_child_count($db_Module);
for ($schild_index = 0; $schild_index < $num_children; $schild_index++)
{
    my $db_Instr = &get_child ($db_Module, $schild_index);
    next unless &get_name ($db_Instr) eq "USER_INSTRUCTION";
    $NIOS_HAS_USER_INSTRUCTIONS="Yes, indeedy";

    my $mnemonic = &get_data ($db_Instr);
    push (@NIOS_USER_MNEMONIC_LIST, $mnemonic);
}

```



```

$NIOS_USER_INSTRUCTION_FORMAT {$mnemonic} =
    &PTF_Get_Required_Data_By_Path ($db_Instr, "format");
$NIOS_USER_INSTRUCTION_OPCODE {$mnemonic} =
    &PTF_Get_Required_Data_By_Path ($db_Instr, "opcode");
$NIOS_USER_INSTRUCTION_GETS_PREFIX {$mnemonic} =
    &PTF_Get_Boolean_Data_By_Path ($db_Instr, "uses_prefix");
$NIOS_USER_INSTRUCTION_IS_SEQUENTIAL {$mnemonic} =
    &PTF_Get_Boolean_Data_By_Path ($db_Instr, "has_sequential_logic");
$NIOS_USER_INSTRUCTION_MODULE_NAME {$mnemonic} =
    &PTF_Get_Required_Data_By_Path ($db_Instr, "top_module_name");
$NIOS_USER_INSTRUCTION_IS_BLACK_BOX {$mnemonic} =
    !&PTF_Get_Boolean_Data_By_Path ($db_Instr, "do_synthesize_top");

$NIOS_USER_INSTRUCTION_SUPPORT_PREFIX |=
    $NIOS_USER_INSTRUCTION_GETS_PREFIX {$mnemonic};

if (! $NIOS_USER_INSTRUCTION_IS_BLACK_BOX {$mnemonic})
{
    my $file_string = &get_data_by_path ($db_Instr,
        "HDL_INFO/synthesis_files");

    push (@Instruction_Unit_Synth_File_List,
        split (/\\s*\\,\\s*/, $file_string));
}
}

# There. We've now set a whole bunch of globals for the Nios
# Vpp-code to use. I'm totally ashamed of myself.

#####
# Firm Flip-Flop Generation
#
# We create several Firm_Flip_Flop variants so that we can, for example,
# assign Fast I/O register attributes to them.
#
# These global variables we're setting get used
# inside the nios-core Vpp script. David Van Brink would be
# horrified.
#
# We assign the "FAST_OUTPUT_REGISTER or "FAST_INPUT_REGISTER"
# attribute to these depending on whether the system-at-hand has a
# "principal" tri-state databus. If it does, then we presume
# off-chip access is speed-critical, and we make these
# assignments. If there is no "principal" tri-state databus, then
# we -explicitly- create the ESF-file anyhow, but with these
# assignments turned OFF. We do this to overwrite any
# past-versions of these esf-files that may be lying around.
#
# These names used to be long and luxurious. Now they're short
# and spartan. FPGA Express gets angry if your name runs over 32
# characters, and the old long names "used up" 16 precious characters.
# Now they only use-up three. But they're short. And spartan. Sorry.
#
$ADDRESS_OUT_REG_MODULE_NAME = $$arg{name} . "_ar"; # _address_out_reg
$CONTROL_OUT_REG_MODULE_NAME = $$arg{name} . "_cr"; # _control_out_reg
$DATA_IN_REG_MODULE_NAME = $$arg{name} . "_dr"; # _data_in_reg

&Create_Firm_Flip_Flop_Variant ($$arg{system_directory},
    $$arg{sopc_directory},
    $$Sys_WSA{device_family},
    $ADDRESS_OUT_REG_MODULE_NAME,
    $CONTROL_OUT_REG_MODULE_NAME,

```

\$DATA_IN_REG_MODULE_NAME
);

my \$fast_switch = (\$\$Sys_WSA{Principal_Tri_State_Data_Bus}) ? "ON" :
"OFF";

&Create_ESF_File ("firm_flip_flop : FAST_OUTPUT_REGISTER = \$fast_switch;",
\$arg{system_directory},
\$ADDRESS_OUT_REG_MODULE_NAME,
\$CONTROL_OUT_REG_MODULE_NAME,
);

&Create_ESF_File ("firm_flip_flop : FAST_INPUT_REGISTER = \$fast_switch;",
\$arg{system_directory},
\$DATA_IN_REG_MODULE_NAME
);

#####

"Run" Vpp

#

The Nios core has more vpp source-files than William Howard Taft
has whiskers. So be it.

#

my \$vpp_source_dir = "\$arg{class_directory}/vpp_source";

my @vpp_file_list = (
"-H", "\$arg{class_directory}/vpp_source/firm_flip_flop.vpp",
"-H", "\$arg{sopc_directory}/bin/vpp/generator_functions.vpp",
"-H", "\$vpp_source_dir/processor_generator_functions.vpp",
"-H", "\$vpp_source_dir/cpu_interface.vpp",
"-H", "\$vpp_source_dir/mnemonics.vpp",
"-H", "\$vpp_source_dir/control_bits.vpp",
"\$vpp_source_dir/major_opcode_table.vpp",
"\$vpp_source_dir/subtable_w.vpp",
"\$vpp_source_dir/instruction_decoder.vpp",
"\$vpp_source_dir/register_ram.vpp",
"\$vpp_source_dir/cpu_core.vpp",
);

if (\$arg{multiply}) {

push (@vpp_file_list, ("vpp_source_dir/mul_unit.vpp"));

push (@Instruction_Unit_Synth_File_List,
"\$arg{system_directory}/\$arg{name}_mul_unit.v");

}

&Progress ("Running Vpp for Nios CPU: \$arg{name}.") if \$arg{verbose};

\$VPP_EXTERNAL_SECURE = 0;

&Vpp (
split (/s+/, "-Q -X v"),
"-R",
"-D", "\$arg{system_directory}",
"-P", \$arg{name} . "_",
"NIOS_DATA_BITS = \$\$SBI{Data_Width} ",
"NIOS_SINGLE_CLOCK_SHIFT_DEPTH = \$arg{shift_size} ",
"NIOS_REGISTER_FILE_SIZE = \$arg{num_regs} ",
"NIOS_WRITEABLE_WVALID_REGISTER = \$arg{wvalid_wr} ",
"NIOS_USE_DECODER_ROMS = \$arg{rom_decoder} ",
"NIOS_MSTEP_SUPPORT = \$arg{mstep} ",
"NIOS_MULTIPLY_SUPPORT = \$arg{multiply} ",
"NIOS_SYSTEM_HIGHEST_ADDRESS = \$highest_address ",
"NIOS_MAIN_MEM_BASE_ADDRESS = \$arg{mm_base} ",
"NIOS_MAIN_MEM_ADDRESS_SPAN = \$arg{mm_span} ",
"NIOS_VECBASE = \$vecbase_address ",
"NIOS_RESET_ADDRESS = \$reset_address "

```

"NIOS_TURNAROUND_IDLE_CYCLE      = $do_use_idle_cycle",
    @globals_2_control_synthesis_place_and_route,
    @vpp_file_list,
);

```

```

#####

```

```

# Convert mif-files to dat-files

```

```

#

```

```

if ($Sys_WSA{do_build_sim} && $$arg{rom_decoder}) {

```

```

    &Nios_Convert_ROM ($arg, "major_opcode_table");

```

```

    &Nios_Convert_ROM ($arg, "subtable_w");

```

```

}

```

```

#####

```

```

# Update the PTF-file so that it has correct

```

```

# ports and HDL-files that agree with the Nios we just

```

```

# generated.

```

```

#

```

```

&Progress ("Updating PTF for Nios CPU: $$arg{name}.") if $$arg{verbose};

```

```

&Add_Module_Ports_To_PTF ($db_Module, $$arg{name});

```

```

&Add_Synthesis_Files_To_PTF ($db_Module, @Instruction_Unit_Synth_File_List,

```

```

    "$$arg{system_directory}/$DATA_IN_REG_MODULE_NAME.v",

```

```

    "$$arg{system_directory}/$ADDRESS_OUT_REG_MODULE_NAME.v",

```

```

    "$$arg{system_directory}/$CONTROL_OUT_REG_MODULE_NAME.v",

```

```

    "$$arg{system_directory}/$$arg{name}\_register_ram.v",

```

```

    "$$arg{system_directory}/$$arg{name}\_major_opcode_table.v",

```

```

    "$$arg{system_directory}/$$arg{name}\_subtable_w.v",

```

```

    "$$arg{system_directory}/$$arg{name}\_instruction_decoder.v",

```

```

    "$$arg{system_directory}/$$arg{name}\_cpu_core.v",

```

```

);

```

```

# We need to list our MIF-files, so their names can be "crushed" if

```

```

# required for MaxPlus+II(two).

```

```

#

```

```

if ($$arg{rom_decoder}) {

```

```

    &Add_HDL_INFO_Files_To_PTF

```

```

        ($db_Module, "MIF_Files",

```

```

            "$$arg{system_directory}/$$arg{name}\_major_opcode_table.mif",

```

```

            "$$arg{system_directory}/$$arg{name}\_subtable_w.mif",

```

```

        );

```

```

}

```

```

&write_ptf_file ($db_PTF_File) or die

```

```

    "Couldn't write PTF File when generating Nios CPU: $$arg{name}";

```

```

}

```

```

#####

```

```

# Nios_Convert_ROM

```

```

#

```

```

# Gets used twice, so I guess it's a subroutine.

```

```

#

```

```

#####

```

```

sub Nios_Convert_ROM

```

```

{

```

```

    my ($arg, $table_name) = (@_);

```

```

    my $full_table_name = "$$arg{name}\_$table_name";

```

```

    &Convert_Mif_To_Dat ("$$arg{system_directory}/$full_table_name.mif",

```

```

        "$$arg{system_sim_dir}/$full_table_name.dat");

```

```

}

```

```

#####

```

```

# fillup_synthesis_place_and_route_globals;

```

098301061061201

25 009880106 1061201

```
#
# Reads values from "SYNTH_CONTROL" section of PTF, builds them
# into a list of name=value pairs.
# This list gets passed along to Vpp, so it can do its evil kludge
5 # of setting global variables.
#
#####
sub fillup_synthesis_place_and_route_globals
{
10   my ($db_Module) = (@_);

   # The trailing "0" argument means "strict=0": It's OK if section doesn't
   # exist
   my $SC_hash = &PTF_Build_Hash_From_Section ($db_Module, "SYNTH_CONTROL", 0);

15   my @name_equals_value_list = ();
   foreach $var_name (keys (%{$SC_hash})) {
       push (@name_equals_value_list, "$var_name=$$SC_hash{$var_name}");
   }
20   return @name_equals_value_list;
}

#####
# Execution begins here
#####
&Mk_Nios (@ARGV);
```